Mob Programming for writing code with high complexity and remote work

Herez Moise Kattan

Text Submitted
to the
Institute of Mathematics and Statistics
of the
University of Sao Paulo
for the
Doctoral Degree
in
Computer Science

Advisor: Prof. Dr. Flavio Soares Correa da Silva

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

Sao Paulo, January 2023

Mob Programming for writing code with high complexity and remote work

This version of the thesis contains the corrections
and suggested changes by the Judging Committee
during the defense of the original version
of the work, held on January 26, 2023.
A copy of the original version is available
at the Institute of Mathematics and Statistics
of the University of Sao Paulo.

Doctoral Thesis Committee:

- Prof. Dr. Flavio Soares Correa da Silva - IME-USP - Department of Computer Science
- Prof. Dr. Xiaofeng Wang - UNIBZ - Faculty of Engineering
- Prof. Dr. Ademar Manuel Teixeira de Aguiar - FEUP - Department of Electrical and Computer Engineering
- Prof. Dr. Graziela Simone Tonin - INSPER - Department of Computer Science
- Prof. Dr. Paulo Sergio Muniz Silva - POLI-USP - Department of Computer and Digital Systems Engineering

The content of this work is published under the CC BY 4.0 license.
(Creative Commons Attribution 4.0 International)

O conteúdo deste trabalho está publicado sob a licença CC BY 4.0.
(Creative Commons Atribuição 4.0 Internacional)

Ficha catalográfica elaborada com dados inseridos pelo(a) autor(a)
Biblioteca Carlos Benjamin de Lyra
Instituto de Matemática e Estatística
Universidade de São Paulo

Kattan, Herez Moise
Mob Programming for writing code with high complexity and remote work / Herez Moise Kattan; orientador, Flávio Soares Corrêa da Silva. - São Paulo, 2023.
284 p.: il.

Tese (Doutorado) - Programa de Pós-Graduação em Ciência da Computação / Instituto de Matemática e Estatística / Universidade de São Paulo.
Bibliografia
Versão corrigida

1. Engenharia de Software. 2. Desenvolvimento de Software. 3. Técnicas de Programação. 4. Qualidade de Software. 5. Resolução de Problemas. I. Silva, Flávio Soares Corrêa da. II. Título.

Bibliotecárias do Serviço de Informação e Biblioteca
Carlos Benjamin de Lyra do IME-USP, responsáveis pela
estrutura de catalogação da publicação de acordo com a AACR2:
Maria Lúcia Ribeiro CRB-8/2766; Stela do Nascimento Madruga CRB 8/7534.

Abstract

KATTAN, H. M. **Mob Programming for writing code with high complexity and remote work**. 2023. 284 p. Thesis (Doctoral) - Institute of Mathematics and Statistcs, University of Sao Paulo, Sao Paulo, 2023.

Nowadays, the world relies on software, and the citizens of this modern world demand good quality software. Moreover, their needs change constantly. Hence, they require new software or new versions of existing ones that correctly solve their problems. Additionally, software development is not just about the cost or quality of software but also about the people who work to produce software. The motivation is to find better ways to organize the work of the programmers to develop quality software in a productive way suitable to their current context. The COVID-19 pandemic impacts the work environment and obligates people to work remotely.

Furthermore, the software developed is increasingly complex. Thus, it emerges in society's need for software development techniques to create complex software with quality and permit software developers to work remotely. Creating reliable, stable, easy maintenance software is to its quality. This research is about an emerging, popular, and current practice to increase the quality of software development, learning, and programmer satisfaction: Mob Programming.

The idea of Mob Programming is the entire team developing software as a group together around a computer and not as an isolated task. The research strategy is a Grounded Theory about Mob Programming. It is observed in practice its benefits in high complexity software development and remote work. Interviews, questionnaires, observations, and source code analysis produce raw data about these observations. Thence is analyzed these data using open code, constant comparison method, core category, selective coding, memoing, sorting, and theoretical coding following the Grounded Theory research method to serve as a recommendation for software developers with suggestions about team roles and practices related to how to use Mob Programming on complex tasks and remote work.

Keywords: Mob Programming, Pair Programming, Simultaneous Style Pair Programming, Software Engineering, Collaborative Programming, Collaborative Problem Solving, Collaboration, Software Quality.

Resumo

KATTAN, H. M. **Mob Programming para escrever código com alta complexidade e trabalho remoto**. 2023. 284 f. Tese (Doutorado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Hoje em dia, o mundo depende de software e os cidadãos deste mundo moderno exigem software de boa qualidade. Além disso, suas necessidades mudam constantemente. Assim, eles demandam um novo software ou novas versões dos existentes que resolvam corretamente seus problemas. Adicionalmente é correto afirmar que o desenvolvimento de software não envolve apenas o custo ou a qualidade do software, mas também as pessoas que trabalham para produzi-lo. A motivação é encontrar melhores formas de organizar o trabalho dos programadores para desenvolver software de qualidade de forma produtiva e adequada ao seu contexto atual. A pandemia COVID-19 impacta o ambiente de trabalho e obriga as pessoas a trabalharem remotamente.

Ademais, o software desenvolvido está cada vez mais complexo. Consequentemente, surge na sociedade a necessidade de técnicas para o desenvolvimento de software complexo com qualidade e que permitam aos desenvolvedores de software trabalharem remotamente. A criação de um software confiável, estável e de fácil manutenção depende de sua qualidade. Esta pesquisa é sobre uma prática emergente, popular e atual para aumentar a qualidade no desenvolvimento de software, aprendizagem e satisfação dos programadores: Mob Programming.

A ideia do Mob Programming é toda a equipe desenvolvendo software como um grupo em torno de um computador e não como uma tarefa isolada. A estratégia de pesquisa é produzir uma Teoria Fundamentada nos Dados observados sobre o Mob Programming. Foram observados na prática os benefícios do Mob Programming no desenvolvimento de software de alta complexidade e trabalho remoto. Entrevistas, questionários, observações e análise do código-fonte produzem dados brutos sobre essas observações. Então, esses dados foram analisados por meio da codificação aberta, análise comparativa constante, categorização, codificação seletiva, elaboração de memorandos, ordenação e, codificação teórica seguindo o método de pesquisa Teoria Fundamentada nos Dados para servir como recomendação para desenvolvedores de software com sugestões sobre funções dos membros da equipe e práticas relacionadas com a forma de usar o Mob Programming em tarefas complexas e trabalho remoto.

Palavras-chave: Programação em Grupo, Programação em Par, Programação e Revisão Simultânea em Pares, Engenharia de Software, Programação Colaborativa, Resolução Colaborativa de Problemas, Colaboração, Qualidade de Software.

Contents

List of Abbreviations xv

List of Figures xvii

List of Tables xxv

1 Introduction 1
- 1.1 Problem Outline . . . 2
- 1.2 Objetive and Research Questions . . . 3
- 1.3 Hypotheses . . . 4
- 1.4 Research Strategy and Research Context . . . 4
- 1.5 Full and Short Papers, Workshop, Doctoral Symposiums, Video, and Posters . . . 13
- 1.6 Claimed Contributions . . . 14
- 1.7 Thesis Structure . . . 19

2 Literature Review 23
- 2.1 Introduction . . . 23
- 2.2 Pair Programming . . . 24
  - 2.2.1 Concept View . . . 24
  - 2.2.2 Definition . . . 25
  - 2.2.3 Helpful to introduce newcomers and to reduce the time-to-benefit but increase the cost . . . 25
  - 2.2.4 Reducing the rate at which developers leave the team, programmer experience, and task complexity . . . 26
  - 2.2.5 Mechanisms to improve the performance of pair programming . . . 26
  - 2.2.6 Confidence, satisfaction, gender, and learning . . . 27
  - 2.2.7 Distributed pair programming or remote pair programming . . . 28
  - 2.2.8 Hybrid pair programming . . . 29
  - 2.2.9 Distributed Pair Programming . . . 29
  - 2.2.10 Camaraderie and personality . . . 29
  - 2.2.11 Dissemination of knowledge, quality, and reduction of defects . . . 30
  - 2.2.12 Small project, many developers and short period of development . . . 31
  - 2.2.13 Informal review, inspection process and communication . . . 31
  - 2.2.14 Keyboard ownership . . . 31
  - 2.2.15 Preceptors of performance and organizational culture . . . 31

2.2.16 Interruptions and cognitive learning . . . 32
2.2.17 Relation with user experience design . . . 32
2.2.18 Helpful for big data/data science projects . . . 33
2.2.19 Productive expert-expert pair . . . 33
2.2.20 Different brain activity was observed between pair programming and solo programming . . . 33
2.2.21 GitHub Copilot vs Human Copilot . . . 34
2.2.22 Summarizing the categories founded in the main topics researched and the main strengths and weakness of Pair Programming . . . 34
2.2.23 Simultaneous Style Pair Programming . . . 36
2.3 Mob Programming . . . 37
2.3.1 Different brain activity was observed between mob programming and solo programming . . . 42
2.3.2 Mob Programming and Pair Induce Higher Levels of Concentration (Attention) Compared to Solo Programming . . . 43
2.3.3 Group Sessions Induce Lower Levels of Calmness (Meditation) Compared to Solo Programming . . . 44
2.3.4 Difficulty Decrease When Working in a Group . . . 44
2.3.5 Online Courses using Mob Programming with a mentor role . . . 44
2.3.6 Remote Mob Programming . . . 45
2.3.7 Mob Programming during COVID-19 . . . 46
2.3.8 Lead time optimization and productivity when applied to complex tasks covering multiple domains . . . 46
2.3.9 Group Size . . . 46
2.3.10 Scaling up to multiple Mob Programming groups . . . 47
2.3.11 Mob Programming Patterns . . . 49
2.3.12 Strengths and Weaknesses of Mob Programming . . . 51
2.4 Complexity . . . 52
2.4.1 Related to Mob Programming . . . 52
2.4.2 Related to Pair Programming . . . 53
2.4.3 Related to complexity management . . . 53
2.5 Information Flow and Knowledge Sharing . . . 54
2.5.1 Patterns of Interaction . . . 55
2.5.2 The number of participating agents . . . 56
2.5.3 If the participating agents know each other . . . 56
2.5.4 The interaction occurs in real-time or asynchronous . . . 56
2.6 Ontology . . . 57
2.6.1 Instantiated Ontology . . . 57

3 Research Design **59**
3.1 Introduction . . . 59
3.2 Literature Review . . . 60
3.2.1 Protocol of the Literature Reviews . . . 61
3.3 Research Persperctive . . . 63

3.4 Observations . . . 63
3.5 Levels of data abstraction in Grounded Theory . . . 64
3.6 Role of the Researcher . . . 65
3.7 Glossary of general Grounded Theory terms . . . 69
3.8 Grounded Theory . . . 70
3.8.1 Roots and Definition . . . 70
3.8.2 Objective . . . 71
3.8.3 Grounded Theory Versions or Schools or Traditions . . . 72
3.8.4 Classical School Grounded Theory life cycle . . . 76
3.8.5 Research Area . . . 77
3.8.6 Minor Literature Review . . . 78
3.8.7 Data Collection . . . 78
3.8.8 Summary of Data Analysis techniques inside Grounded Theory . . . 84
3.8.9 Definitions of Codes and Coding . . . 85
3.8.10 Open Coding . . . 85
3.8.11 Constant Comparison Method . . . 85
3.8.12 Selective Coding . . . 86
3.8.13 Memoing . . . 86
3.8.14 Sorting . . . 87
3.8.15 Major Literature Review . . . 87
3.8.16 Theoretical Coding . . . 88
3.8.17 Evaluating a Grounded Theory . . . 89
3.8.18 Limitations . . . 90
3.9 Discussion . . . 90
3.10 Conclusion . . . 91

4 Mob Programming Team Roles Ontology 93
4.1 Introduction . . . 94
4.2 Driver or Pilot . . . 95
4.3 Navigator or Co-Pilot . . . 97
4.4 Facilitator or Mentor . . . 100
4.5 Researcher or Scout . . . 102
4.6 Quality Analyst or Quality Guardian . . . 104
4.7 Nose or Housekeeper . . . 105
4.8 CyberSecurity Analyst or Hacker . . . 106
4.9 Discussion . . . 107
4.9.1 Quality Map . . . 107
4.9.2 Team Roles . . . 107
4.9.3 Team Strategies toward a good performance . . . 108
4.10 Well-Defined Team Roles Theory . . . 108
4.10.1 Context . . . 110
4.10.2 Conditions . . . 110
4.10.3 Causes . . . 110
4.10.4 Consequences . . . 110

4.10.5 Contingencies . . . 110
4.10.6 Covariances . . . 110
4.11 Conclusion . . . 111

5 An Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors 113
5.1 Introduction . . . 114
5.2 Communication Space . . . 114
5.2.1 Productive dialogue . . . 115
5.3 Driver rotation techniques, as Facilitators for productive dialogues . . . 116
5.3.1 Facilitator . . . 116
5.3.2 Initial Information System . . . 116
5.3.3 Standardzed Procedures . . . 117
5.4 Team Members Rotation among the distinct Mob Programming sessions . . . 117
5.5 How many programmers in a session of Mob Programming . . . 117
5.6 Remove one person and team performance . . . 117
5.7 Remove more than one person and team performance . . . 117
5.8 Retrospectives . . . 118
5.9 Structure breaks . . . 118
5.10 Programmer Satisfaction . . . 118
5.11 Kind and Fearlessness Programmer . . . 119
5.12 Room Setup . . . 119
5.12.1 Real-Time Communications . . . 120
5.13 Software setup . . . 122
5.14 Automation . . . 122
5.15 Practices related to the COVID-19 Pandemic . . . 122
5.15.1 Implement Workplace Controls . . . 123
5.15.2 Safe Work Practices . . . 124
5.15.3 Personal Protective Equipment (PPE) . . . 124
5.15.4 Using Mob Programming in Post-Pandemic . . . 125
5.16 Discussion . . . 125
5.16.1 Group Size . . . 128
5.17 Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable . . . 129
5.17.1 Context . . . 131
5.17.2 Conditions . . . 131
5.17.3 Causes . . . 131
5.17.4 Consequences . . . 131
5.17.5 Contingencies . . . 131
5.17.6 Covariances . . . 131
5.18 Conclusion . . . 132

6 Challenges to Start Adopting Mob Programming 135
6.1 Convincing managers and co-workers . . . 135
6.2 Keeping the Focus . . . 136
6.3 Different skill levels . . . 136
6.4 Mental Exhaustion . . . 137
6.5 Intense collaboration can be hard . . . 138
6.6 Interruptions by meetings . . . 139
6.7 Embracing the Vulnerabilities with Kindness . . . 139
6.8 No time for yourself . . . 140
6.9 Discussion . . . 141
6.10 Embracing Vulnerabilities with Kindness Theory . . . 141
6.10.1 Context . . . 143
6.10.2 Conditions . . . 143
6.10.3 Causes . . . 143
6.10.4 Consequences . . . 143
6.10.5 Contingencies . . . 143
6.10.6 Covariances . . . 143

7 Findings Related to Managing the Complexity 145
7.1 Introduction . . . 145
7.2 Sharing operational and domain problem knowledge . . . 146
7.3 Quality . . . 147
7.4 Programmers Satisfaction . . . 147
7.5 Programming of complex and critical code . . . 147
7.6 Collective Code Ownership . . . 148
7.7 Work in Progress Limit . . . 148
7.8 More than two pairs of eyes doing an instant Code review and Refactoring . . . 148
7.9 Knowledge Sharing . . . 149
7.10 Productive Dialogues about Source-Code Quality and Deliver Value . . . 149
7.11 Keeping focus . . . 149
7.12 Fast onboarding of new team members . . . 150
7.13 More modes of thinking combined . . . 150
7.14 Discussion . . . 151
7.14.1 Reducing the rate at which developers leave the team . . . 151
7.15 Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming . . . 151
7.15.1 Context . . . 153
7.15.2 Conditions . . . 153
7.15.3 Causes . . . 153
7.15.4 Consequences . . . 153
7.15.5 Contingencies . . . 153
7.15.6 Covariances . . . 153

8 Results About Remote Work **155**
8.1 Introduction . . . 155
8.2 Software setup . . . 156
8.3 Driver rotation techniques . . . 157
8.4 Retrospectives . . . 158
8.5 Communication tools . . . 158
8.6 Automation . . . 158
8.7 Structure breaks . . . 159
8.8 How many programmers in a session of Mob Programming . . . 159
8.9 Remove one person and team performance . . . 160
8.10 Programmer Satisfaction . . . 160
8.11 Benefits in long term . . . 160
8.12 A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage . . . 161
8.12.1 Context . . . 163
8.12.2 Conditions . . . 163
8.12.3 Causes . . . 163
8.12.4 Consequences . . . 163
8.12.5 Contingencies . . . 163
8.12.6 Covariances . . . 163
8.13 Theory of Benefits of Remote Mob Programming . . . 163
8.13.1 Context . . . 165
8.13.2 Conditions . . . 165
8.13.3 Causes . . . 165
8.13.4 Consequences . . . 165
8.13.5 Contingencies . . . 165
8.13.6 Covariances . . . 165

9 Conclusion **167**
9.1 Research Contributions . . . 167
9.2 Mob Programming Team Roles . . . 168
9.3 Mob Programming Practices, Mob Programmers' Behaviors, and Mob Programming Communication Space . . . 168
9.4 Challenges to Start Adopting Mob Programming . . . 169
9.5 Roles-Practices-Behaviors-Communication Space relationships related to Complexity 170
9.6 Limitations . . . 170
9.6.1 Number of Participants . . . 171
9.6.2 Ontologies . . . 171
9.6.3 Pioneer . . . 171
9.7 Discussion . . . 171
9.7.1 Team Roles . . . 171
9.7.2 Team Practices and Team Members' Behaviours . . . 172
9.7.3 Evaluating a Grounded Theory . . . 173
9.8 Implications for Practice . . . 173

9.8.1 Implications for Teams . . . 173
9.8.2 Implications for Senior Management . . . 174
9.9 Implications for Academia . . . 174
9.10 Future Work . . . 177

A Questionnaire in Portuguese **179**

B Questionnaire in English **185**

C Interview Script in Portuguese **191**

D Interview Script in English **193**

E TCLE **195**

F Informed Consent Document in English **199**

G Authorizations of the Ethics Committees **203**

H Institution Official Consent Terms **211**
H.1 Written Approval of the Rocket.Chat . . . 211
H.2 Written Approval of the Brazilian Air Force . . . 212
H.3 Written Approval of the Hunter Industries . . . 213
H.4 Written Approval of the Mob Mentality . . . 214
H.5 Written Approval of the Shift . . . 215
H.6 Written Approval of the CoteiBem . . . 216
H.7 Summary of Digital Approval of the CoteiBem . . . 217
H.8 Written Approval of the SindicoNet . . . 218

I Background of the Researcher **219**
I.1 Paula Souza State Center for Technological Education . . . 219
I.2 Institute for Technological Research . . . 223
I.3 Institute of Mathematics and Statistics . . . 226
I.4 Certificates of Courses . . . 231
I.5 Microsoft . . . 240
I.6 Delphi . . . 243
I.7 Author . . . 244
I.8 TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research . . . 248
I.9 Diploma of Honor to Merit from the city of Sao Paulo . . . 254
I.10 The First Letter of Saint Paul to the Corinthians . . . 255

Bibliography **257**

Index **281**

List of Abbreviations

PHPMD	PHP Mess Detector (*Detector de Bagunça PHP*)
OAuth	Open Authorization (*Padrão Aberto para Autorização*)
PRSP	Programming and Review Simultaneous in Pairs (*Programação e revisão simultânea em Pares*)
LDAP	Lightweight Directory Access Protocol (*Protocolo Leve de Acesso a Diretório*)
E2EE	End-to-End Encryption (*Criptografia de Ponta a Ponta*)
SCMs	Source Code Management systems (*Sistemas de Gerenciamento de Código-fonte*)
NGOs	Non-Governmental Organizations (*Organizações Não Governamentais*)
CEO	Chief Executive Officer (*Diretor Executivo*)
USD	United States Dollar (*Dólar dos Estados Unidos*)
NSF	National Science Foundation (*Fundação Nacional da Ciência*)
URL	Uniform Resource Locator (*Localizador Uniforme de Recursos*)
SVN	Subversion (*Subversion*)
GIT	Global information tracker (*Rastreamento global de informações*)
PHP	PHP: Hypertext Preprocessor (*PHP: Pré-processador de Hipertexto*)
2FA	Two-Factor Authentication (*Autenticação de Dois Fatores*)
SSO	Single Sign-On (*Login Único*)
LAN	Local Area Network (*Rede Local*)
ATDD	Acceptance Test Driven Development (*Desenvolvimento Orientado a Teste de Aceitação*)
BDD	Behavior Driven Development (*Desenvolvimento Guiado por Comportamento*)
TDD	Test Driven Development (*Desenvolvimento Orientado a Teste*)
IDE	Integrated development environment (*Ambiente de desenvolvimento integrado*)
XP	Extreme Programming (*Programação Extrema*)
GT	Grounded Theory (*Teoria Fundamentada*)
RQ	Research Question (*Questão de Pesquisa*)
CC	Claimed Contribution (*Alegação de Contribuição*)
H	Hypothesis (*Hipótese*)
CEP	Ethics Committee (*Comitês de Ética em Pesquisa*)
CONEP	National Commission of Ethics in Research (*Comissão Nacional de Ética em Pesquisa*)
CNS	National Health Council (*Conselho Nacional de Saúde*)
SUS	Unified Health System (*Sistema Único de Saúde*)
CCA	Brazilian Aeronautics Computing Center (*Centro De Computação Da Aeronáutica*)
SJ	Sao Jose dos Campos (*São José Dos Campos*)
FAB	Brazilian Air Force (*Força Aérea Brasileira*)
USAF	United States Air Force (*Força Aérea dos Estados Unidos*)

List of Figures

1.1 Timeline of the experiences of more than 18 years of the existence of the eXtreme Programming Laboratory course. 5

1.2 Author Fab_oficial. A-1M at Campo Grande Air Base, Brazil, during EXCON Tapio 2022. Instagram, August 30 of 2022. 10

1.3 Author AeronauticaOficial. GRIPEN replica in exposition at BarraShoppingSul, in the city of Porto Alegre in Brazil. Facebook, September 16 of 2022. 11

1.4 Author Anapolis Air Base (BAAN). A delegation from the 1st Air Defense Group and technicians from the Logistic Group at Anapolis Air Base (BAAN) visited the Gripen Design and Development Network and Gripen Flight Test Center to exchange knowledge between SAAB technicians and future pilots, maintainers, and system operators of the new fighter jet of the Brazilian Air Force, in addition to acting in the development of Gripen evolution software worldwide, in the city of Anapolis in Brazil. Fab.mil.br, Edition: Air Force Agency - Review: Major Oliveira Lima. November 10 of 2021. 12

2.1 An illustration of Menzies *et al.* (2003) about pair programming as a central practice. Observe in the diagram pair programming is directly related to almost all the other techniques of the eXtreme Programming from Beck (2000). 25

2.2 A cognitive map summarizing the categories founded in the main topics researched about Pair Programming presented in subsections 2.2.1 until 2.2.21 34

2.3 Drawing about the five phases inside of the Programming and Review Simultaneous in Pairs. The total is seven phases, the first for planning and the last for productive rest are not at drawing. 37

2.4 This setup has been found to work well and has been seen at several companies. You can have one, two, or more monitors. Experiment with monitor height to make sure it is at a height that works well for everyone, illustration from Zuill and Meadows (2016). 39

2.5 This is a Zuill and Meadows (2016) illustration of a room setup that permits the Driver to see the screen simultaneously while the other Mob Programmers watch on a large monitor. 40

2.6 The team is working on the programming. Note the Navigator is strategically positioned, and he has a laser pointer, making it easier to point out an issue in the source code to be fixed, photo from Kattan *et al.* (2018a). 41

2.7 The team could use the imagination because in the fundamentals of a Mob Programming is valued the liberty of choice according to Zuill and Meadows (2016). 42
2.8 Comparison of meditation and concentration (attention) levels by programming state from Shiraishi *et al.* (2021). 43
2.9 Box and whisker of participants' average percentage in change of concentration (attention), meditation, and difficulty from the neutral state by programming session from Comparison of meditation and attention levels by programming state from Shiraishi *et al.* (2021). 43
2.10 Timeline and Forcefield Retrospective for Bi-Annual Goal Setting. The time line yields insights into events that were positive and negative for the developer and the force field retrospective provides insight into every day behaviors or elements of that person's life that are contributing to or taking away from that person's work experience. from Lucian (2017). 48
2.11 Pattern map summarizing the teams' mob programming patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019). 49
2.12 Maps as instantiated ontologies for the interaction between a cartographer and a map reader da Silva and Agusti-Cullell (2008). 54
2.13 Maps as instantiated ontologies for the interaction among a cartographer, a tourist, and a local citizen da Silva and Agusti-Cullell (2008). 55

3.1 Levels of data abstraction in Grounded Theory 64
3.2 TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research in the original box. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the some books Isaaman *et al.* (1982), ISaaman and Tyler (1982), Waters *et al.* (1983). 65
3.3 TK95 Color Computer - Original Box - Back View. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the box original of TK 95, a ZX Spectrum Brazilian Clone, working on TV, with three peripherals, the recorder to save the data in cassette tapes, the matrix printer, and the lighting pen to draw on the screen. 66
3.4 TK95 Color Computer Keyboard. Note the BASIC Language commands are in the keys. 66
3.5 TK95 Color Computer inside the Original Box. Note a cassette tape on the left superior corner. It was the method to persist your data. On the one hand, it is limited. On the other hand, forty years later, the data are stored with no one sector damaged, and the software runs on the computer without compatibility issues. Moreover, it is years of light ahead of modern computers in energy saving, which is crucial nowadays because of climate change. This is a treasure to be studied. 67
3.6 Front view of the AC power adapter of the TK95 Color Computer. 67
3.7 TK95 Color Computer - Back View. The first connector on the right is to plug the TV. The others in the sequence are to connect the earphones, tape recorder, and microphone. Following are the connectors for the joystick and the next one for peripherals, e. g., the lighting pen shown in the back view of the box. The last is to connect the AC power adapter. 68

3.8 The Coding Procedure of the Classical version of Grounded Theory Holton (2010) and Kenny and Fourie (2015). 72
3.9 The Coding Procedure of the Straussian Grounded Theory Strauss and Corbin (1990) and Kenny and Fourie (2015). 73
3.10 The Coding Procedure of the Constructivist version of Grounded Theory Charmaz (2008) and Kenny and Fourie (2015). 74
3.11 The Uniting and Differentiating Principles of the distinct Grounded Theory versions Kenny and Fourie (2015). 76
3.12 Overview of Grounded Theory research method image adapted from Hoda *et al.* (2012). The figure was included inside the rectangle Research Area. 77
3.13 Memo used in the Mob Programming Team Roles Ontology to the Role of the Driver. Photo by mohamedhassan from Freerange Stock. The image was adapted with the text of the Memo. 86
3.14 Using Hoda *et al.* (2011) as an example that guided us in the development of our Six C's model, the title of her paper is "the impact of inadequate customer involvement on self-organizing Agile teams", in terms of the Six C's theoretical model for the category Lack of Customer Involvement. Using the Six C's model, Hoda *et al.* (2011) describe (1) Contexts: the ambiance, that is, the context of the Agile development teams in NZ and India, (2) Conditions: factors that are prerequisites for the category, Lack of Customer Involvement, to manifest, (3) Causes: reasons that cause lack of customer involvement, (4) Consequences: outcomes or effects of lack of customer involvement on self-organizing Agile teams, (5) Contingencies: moderating factors between causes and consequences, that is, Agile Undercover strategies, and (6) Covariances: correlations between different categories, that is, how Agile Undercover strategies change when factors that cause Lack of Customer Involvement change. The theory of Lack of Customer Involvement depicted using the Six C's model (Context, Condition, Causes, Consequences, Contingencies, and Covariance). 89

4.1 Mob Programming team roles emerged through grounded theory, more specifically in the analysis of underlying categories and concepts. 94
4.2 Key points about the Driver/Pilot role emerged through grounded theory when investigating underlying categories and concepts. 96
4.3 Key points about the Navigator/Co-Pilot role emerged through grounded theory when investigating underlying categories and concepts. 98
4.4 Key points about the Facilitator role emerged through grounded theory when investigating underlying categories and concepts. 100
4.5 Key points about the Researcher role emerged through grounded theory when investigating underlying categories and concepts. 102
4.6 Quality Analyst team role, responsible for the creation of a Code Quality Map. Map Photo by Jmohamedhassan from Freerange Stock. 104
4.7 Nose team role, responsible for the creation of the Code Smells Map. Map Photo by Jack Moreh from Freerange Stock. 105

4.8 CyberSecurity Analyst team role. It is responsible for the creation of Security Lists and Hardening CheckLists. Computer with a PadLock, Photo by Jack Moreh from Freerange Stock. Map Photo by BlueOrange from Freerange Stock . . . 106
4.9 Six C's model of Well-Defined Team Roles Theory. . . . 109

5.1 Collaborative construction of a communication space from da Silva and Agusti-Cullell (2008). . . . 114
5.2 K6 complete . . . 119
5.3 K7 complete . . . 120
5.4 Voice. Sound Waves. Photo of the Left by StuartMiles from Freerange Stock. Photo of the Right by Gratisography / Ryan McGuire from Freerange Stock . . . 121
5.5 Voice. Sound Waves. Photo by StuartMiles from Freerange Stock . . . 121
5.6 Planar complete graph K4. . . . 127
5.7 Another possibility of to draw the Planar complete graph K4. It is a triangle that does not cross any of your edges, and its lines are straight. . . . 128
5.8 Complete graph K5 is not planar. Note it is impossible to draw K5 without crossing any edges. On the other hand, observe that K2, K3, and K4 are not cross their lines. Therefore, they are planar. . . . 129
5.9 Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable . . . 130

6.1 Embracing Vulnerabilities with Kindness Theory . . . 142

7.1 Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming. . . . 152

8.1 A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors. . . . 162
8.2 Theory of Benefits of Remote Mob Programming . . . 164

A.1 First Page of the Questionnaire in Portuguese . . . 179
A.2 Second Page of the Questionnaire in Portuguese . . . 180
A.3 Third Page of the Questionnaire in Portuguese . . . 181
A.4 Fourth Page of the Questionnaire in Portuguese . . . 182
A.5 Fifth Page of the Questionnaire in Portuguese . . . 183

B.1 First Page of the Questionnaire in English . . . 185
B.2 Second Page of the Questionnaire in English . . . 186
B.3 Third Page of the Questionnaire in English . . . 187
B.4 Fourth Page of the Questionnaire in English . . . 188
B.5 Fifth Page of the Questionnaire in English . . . 189

C.1 First Page of the Interview Script in Portuguese . . . 191
C.2 Second Page of the Interview Script in Portuguese . . . 192

D.1 First Page of the Interview Script in English . . . 193
D.2 Second Page of the Interview Script in English . . . 194

E.1 First Page of the Termo de Consentimento Livre e Esclarecido (TCLE) . . . 195
E.2 Second Page of the Termo de Consentimento Livre e Esclarecido (TCLE) . . . 196
E.3 Third Page of the Termo de Consentimento Livre e Esclarecido (TCLE) . . . 197

F.1 First Page of the Informed Consent Document in English . . . 199
F.2 Second Page of the Informed Consent Document in English . . . 200
F.3 Third Page of the Informed Consent Document in English . . . 201

G.1 Authorization of the Ethics Committee of the Institute of Mathematics and Statistics of the University of Sao Paulo. . . . 203
G.2 Solicitation to the Research Committee of the Institute of Mathematics and Statistics of the University of Sao Paulo. . . . 204
G.3 Authorization from the IME-USP Research Committee and need to register in the Plataforma Brasil. . . . 205
G.4 The cover sheet of this research is in the registry of *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP). . . . 206
G.5 Page 1 of the Written Approval of the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP). . . . 207
G.6 Page 2 of the Written Approval of the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP). . . . 208
G.7 Page 3 of the Written Approval of the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP). . . . 209
G.8 Page 4 of the Written Approval of the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP). . . . 210

H.1 Autorization of the Rocket.Chat . . . 211
H.2 Autorization of the Brazilian Air Force . . . 212
H.3 Autorization of the Hunter Industries . . . 213
H.4 Autorization of the Mob Mentality . . . 214
H.5 Autorization of the Shift . . . 215
H.6 Autorization of the CoteiBem . . . 216
H.7 Summary of Digital Approval of the CoteiBem . . . 217
H.8 Autorization of the SindicoNet . . . 218

I.1 Full Professional Qualification in Data Processing Technician by the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 219
I.2 School Record of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 220

I.3 Page one of Internship Assessment of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 221
I.4 Page two of Internship Assessment of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 222
I.5 Master of Science in Computer Engineering . . . 223
I.6 School Record from the Institute for Technological Research of the State of Sao Paulo (IPT) - page one. . . . 224
I.7 School Record from the Institute for Technological Research of the State of Sao Paulo (IPT) - page two. . . . 225
I.8 School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page one. . . . 226
I.9 School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page two. . . . 227
I.10 School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page three. . . . 228
I.11 Certificate working as a moderator in the Science and Technology Week from the University of Sao Paulo. . . . 229
I.12 Recommendation letter from FLOSS Competence Center of USP. . . . 230
I.13 Certificate of Course in Programming, in an early stage of my career as a teenager. Also, I put another from my childhood regarding the programming course card in subsection 3.6 "Role of the Researcher". . . . 231
I.14 Certificate of the ISO 9000 course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 232
I.15 Certificate of the Visual Basic course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP). . . . 233
I.16 Certificate of the Visual Interdev course from the Microsoft authorized technical education center G&P linked to Microsoft Official Curriculum. . . . 234
I.17 Certificate of the Windows NT Core Technologies course from the Microsoft authorized technical education center G&P linked to Microsoft Official Curriculum. . . . 235
I.18 Certification of the Managing the Classroom Environment from the Microsoft authorized technical education center Impacta linked to Microsoft Official Curriculum. . . . 236
I.19 Certificate of the FIX Protocol course from the Sao Paulo Stock Exchange and the Brazilian Mercantile Futures Exchange, now called B3, is linked to CME Group. . . . 237
I.20 Certificate of the Introduction to Citizen Science focus on biodiversity course from USP and University of Massachusetts. . . . 238
I.21 Certificate of the Network course. . . . 239
I.22 Letter from Microsoft to the United States of America Consulate General in Brazil for the researcher author of this thesis to have a Visa. . . . 240
I.23 Examination Score Report Microsoft' Designing and Implementing Desktop Applications. . . . 241

I.24 Examination Score Report Microsoft' Designing and Implementing Distributed Applications. . . . 242
I.25 Recommendation letter from Delphi. . . . 243
I.26 Register Number 265.065 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998. . . . 244
I.27 Register Number 265.066 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998. . . . 245
I.28 Register Number 265.067 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998. . . . 246
I.29 Register Number 265.068 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998. . . . 247
I.30 TK95 Color Computer - Original Box - Front View. . . . 248
I.31 TK95 Color Computer - Original Box - Lateral View. Note the circle, compatilhe with ZX Spectrum. . . . 248
I.32 TK95 Color Computer - Original Box - Lateral View. Note that one side is written in Portuguese, but the other is written in Spanish because Brazil exports to Latin America. . . . 249
I.33 TK95 Color Computer - Original Box - Back View. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the box original of TK 95, a ZX Spectrum Brazilian Clone, working on TV, with three peripherals, the recorder to save the data in cassette tapes, the matrix printer, and the lighting pen to draw on the screen. . . . 249
I.34 Original styrofoam box, it helped preserve the computer for 40 years. But the paper of the original instructions book has suffered over time. . . . 250
I.35 Software stored in the cassette tape. They have two sides. The data are flawlessly stored after 40 years. . . . 250

I.36 Lado B of the cassette tape. They have two sides. The data are flawlessly stored after 40 years. 251
I.37 TK95 Color Computer inside the Original Box. Note a cassette tape on the left superior corner. It was the method to persist your data. On the one hand, it is limited. On the other hand, forty years later, the data are stored with no one sector damaged, and the software runs on the computer without compatibility issues. Moreover, it is years of light ahead of modern computers in energy saving, which is crucial nowadays because of climate change. This is a treasure to be studied. 251
I.38 TK95 Color Computer Keyboard. Note the BASIC Language commands are in the keys. 252
I.39 Front view of the AC power adapter of the TK95 Color Computer. 252
I.40 TK95 Color Computer - Back View. The first connector on the right is to plug the TV. The others in the sequence are to connect the earphones, tape recorder, and microphone. Following are the connectors for the joystick and the next one for peripherals, e. g., the lighting pen shown in the back view of the box. The last is to connect the AC power adapter. 253
I.41 TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research in the original box. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the some books Isaaman *et al.* (1982), ISaaman and Tyler (1982), Waters *et al.* (1983). 253
I.42 Diploma of Honor to Merit from the city of Sao Paulo signed by the epoch Mayor Mario Covas[1], who after became the majority leader of the Constitutional Assembly from Brazil, which drew up a new democratic charter. 254
I.43 Author: Mary Harrsch, Title: St Paul outside St Peter's Basilica in Vatican City. Photographed in St. Peter's Basilica in Vatican City. Available at: Flickr.com in her profile MHarrsch in the album The Vatican under the Creative Commons license: Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0). 255

List of Tables

1.1 Relation of the proximity of the Claimed Contributions with the Researchers in an academic environment or with software developers in the industry of software or with both of them. The letter "X" means the connection of the Claimed Contribution with Researchers, Practitioners, or both of them. 15

1.2 This table contains only a short description of the two Ontologies since more details beyond this brief description of Mob Programming Team Roles Ontology are available in Chapter 4, and about the Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors in Chapter 5. 17

1.3 This table contains only a short description of the first four Theories since more details beyond this brief description of Well-Defined Team Roles Theory are available in Section 4.10, and about the Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable in Section 5.17, and about Embracing Vulnerabilities with Kindness Theory, in Section 6.10, and about the Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming in Section 7.15. 18

1.4 This table contains only a short description of the last two Theories since more details beyond this brief description about a theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage are available in Section 8.12, and about the Theory of Benefits of Remote Mob Programming are available in Section 8.13. 19

2.1 Guideline for when to use pair programming of Dyba *et al.* (2007) 26

2.2 The main strengths and weakness of Pair Programming presented in subsections 2.2.1 until 2.2.21 in a chronological way 35

2.3 Summary the teams' mob programming roles patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019). 50

2.4 Summary the teams' mob programming collaboration patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019). 50

2.5 Summary the teams' mob programming driving patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019). 51

2.6 Strengths and weaknesses of the Mob Programming 51

3.1 Inclusion and exclusion criteria of a paper to be accepted here in this systematic literature review about Pair Programming according to the guideline of Kitchenham *et al.* (2004). 62

3.2 Inclusion and exclusion criteria of a paper to be accepted here in this systematic literature review about Mob Programming according to the guideline of Kitchenham *et al.* (2004). 63
3.3 Part one of a glossary of general Grounded Theory (GT) terms adapted from Hoda *et al.* (2010) and Hoda *et al.* (2012). 69
3.4 Part two of a glossary of general Grounded Theory (GT) terms adapted from Hoda *et al.* (2010) and Hoda *et al.* (2012). 70
3.5 The reliability is measured using the Internal Consistency strategy, the evaluation of the probability of the survey to provide consistent results when replicated in identical conditions. The value of the Cronbach's alpha correlating with the respective classification of the level of reliability in the Multiple-item Measurements. 81
3.6 Open Source Software used in the Interviews. 83
3.7 Short descriptions of the data analysis techniques inside GT (the "classical" version) used by this thesis. The procedure of data analysis called coding in GT. There are two types of coding: Substantive coding and Theoretical coding, In this table, are summarized the coding mechanisms: Open Coding and Selective Coding, that lead to substantive codes; Memoing and sorting; and Theoretical coding. 84
3.8 Software engineering research techniques, methods, and approaches used in this research and their respective considerations. 91

4.1 Mob Programming Team Roles of the Driver or Pilot, with its interactions in real-time at Mob Programming session among the team members. 95
4.2 Mob Programming Team Roles of the Driver or Pilot, with its interactions correlation if they know each other and the programming language. 97
4.3 Mob Programming Team Roles of the Navigator or Co-Pilot, with its interactions in real-time at Mob Programming session among the team members. 99
4.4 Mob Programming Team Roles of the Navigator or Co-Pilot, with its interactions correlation if they know each other and the programming language. 99
4.5 Mob Programming Team Roles of the Researcher or Scout, with its interactions in real-time at Mob Programming session among the team members. 103

9.1 This table contains only a short description of the two Ontologies since more details beyond this brief description of Mob Programming Team Roles Ontology are available in Chapter 4, and about the Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors in Chapter 5. 174
9.2 This table contains only a short description of the first four Theories since more details beyond this brief description of Well-Defined Team Roles Theory are available in Section 4.10, and about the Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable in Section 5.17, and about Embracing Vulnerabilities with Kindness Theory, in Section 6.10, and about the Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming in Section 7.15. 175

9.3 This table contains only a short description of the last two Theories since more details beyond this brief description about a theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage are available in Section 8.12, and about the Theory of Benefits of Remote Mob Programming are available in Section 8.13. 176

Chapter 1

Introduction

The motivation is to find better ways to organize the work of the programmers to develop quality software in a productive way suitable to their current context. Because the present world depends on software, the citizens of this modern world want to use good quality software. Moreover, their needs change rapidly. Thus, they lack new software or versions of existing ones that correctly solve those problems as soon as possible. Additionally, software development is not just about the cost or quality of software but also about the people who work to produce software. Therefore, the professionals involved also have their needs, e.g., a safe and innovative environment to develop themselves.

The social isolation resulting from the COVID-19 pandemic, also known as the coronavirus pandemic, impacts the work environment and obligates people to work remotely. This social isolation is a potential stress factor. Moreover, the software developed is growing in complexity. Thus, it emerges in society's need for software development techniques to create complex software with quality and permit software developers to work remotely, taking into account the well-being of the programmers.

This research project is the continuation of the master thesis of Kattan (2015), which intends to provide experiments to explore the most suitable forms to allow programmers to develop software productively considering to improve the satisfaction of the programmers and their learning experience, including remotely. Primarily, it investigated a new style of Pair Programming. Pair programming is a practice for software development teams, and it is also a process of informal review [Williams *et al.* (2000)] [Cockburn and Williams (2000)] [Cockburn and Williams (2001)]. Work in pairs improves the software quality [di Bella *et al.* (2013)]. It increases the technical level of the team [Kattan (2018b)] [Kattan *et al.* (2018b)] [Kattan (2019a)] because of the knowledge sharing [Williams and Kessler (2000)] [Williams and Kessler (2003)] [Plonka *et al.* (2015)].

The programmers sitting side by side and using the same computer have distinct roles. These roles are respectively called pilot and co-pilot or driver and navigator. The pilot/driver writes the code, thinks in algorithms. The co-pilot/navigator observes the code produced, tries to think about how to improve the algorithm. Furthermore, he also checks for possible errors and points of failure [Begel and Nagappan (2008)]. The idea is to develop software in pairs and not as an isolated task. This approach promotes up social aspects of programming. While one developer writes the algorithm, the partner can do code review, discuss alternatives, check the quality of code, or help in complex regions [Coplien and Schmidt (1995)].

Mob Programming is a new approach, and haven't elapsed time enough to appear in publications that studied to know and discover many details scientifically about it. Thus, pair programming was studied more. Furthermore, pair programming has similarities with Mob Programming. Due to Mob Programming seems to be a super pairing, we list some pair programming advantages beyond the increase of quality perceived are:

1. Pair programming involves constant communication and collaboration, encouraging the dissemination of knowledge among the team members. Pair programming fosters confidence among developers [Constantine (1995)];

2. The defects decrease working in pairs compared to solo programming [Nosek (1998)];

3. Pair programming encourages companionship. It is a protective factor for keeping computer science students [McDowell *et al.* (2006)]. It decreases the dropout of women and novice programmers in computer science courses. Thus, it could be a low-cost approach for reducing the turn-over in software teams;

4. The partner, who does not own the keyboard, contributes with good ideas [Williams and Kessler (2003)] this helps to maintain discipline and concentration in productive activities [Kattan (2018c)].

Pair programming has generally produced software with higher quality compared with those developed through solo programming. However, the literature review of pair programming revealed some drawbacks, such as a loss of its benefits in some cases specifically, it does not work thoroughly for senior programmers, and simple tasks [Arisholm *et al.* (2007)] [Dyba *et al.* (2007)]. Pair programming might be more useful only in complex tasks [Hulkko and Abrahamsson (2005)] [Vanhanen *et al.* (2007)]. It reported when using pair programming increases the effort in the project [Nosek (1998)]. Effort increase is a common criticism of pair programming since it devotes two professionals to perform the same task on a single computer. There is a constant debate in academic and professional circles if the advantages justify these possible productivity impacts and implications [Allen Parrish and Hale (2004)].

The related techniques, such as Mob Programming, where the whole team works around a single computer [Kattan *et al.* (2018a)] [Kattan (2019b)], or code review, another well-known alternative, have advantages compared to traditional-style pair programming. E.g., it delivers software quicker than pair programming [Rajendran Swamidurai and Kannan (2014)]. However, they do not have exactly all the same pair programming benefits. What suggests that with more related studies, we could discover more details from each of these approaches toward better knowing how each one works and making a better choice for each specific context.

The goal of this doctoral thesis is carried out in practice a broader study in the context of Mob Programming. This thesis presents an empirical observation where the raw data are analyzed to emerge one theory using the classical version of the Grounded Theory research method.

This chapter briefly presents the problem outline, the research strategy, and the general context. It also describes the objective, research questions, hypotheses, research context, publications list, and claimed contributions. Finally, we[1] describes the thesis structure with a short explanation of the chapters.

1.1 Problem Outline

The problem outline is to study, observe and verify Mob Programming in practice. Thus, we could deepen the knowledge relative to this emergent practice. Because their advocates argue about Mob Programming is good for increasing software development quality. Moreover, it helps in the software developers' learning and satisfaction.

Software development product with long life and easy maintenance is related to the quality of the software product. Then, society could get the benefits of better quality software, painless to maintain, and without forgetting the due consideration to the professionals involved in software development, because they need a good work environment in which an approach such as Mob Programming could contribute to their learning and satisfaction.

Beyond software quality and programmers' learning and satisfaction, the problem outline is discovering if Mob Programming is an efficient technique for writing highly complex codes. It would

[1]We adopted the following writing style, often using "we" to describe what was done in the research because it is a collaborative process with my supervisor and other colleagues and researchers who provide valuable feedback. Also, other times use "I" to express a particular opinion or a part done only by myself.

help the practitioners choose Mob Programming instead of pair programming when programming software with high complexity.

Finally, whether we like it or not, adapting was the only option, and without alternatives, we included Remote Work in this research. It was needed to adapt to the COVID-19 pandemic, which expanded the problem outline to include Work remotely.

1.2 Objetive and Research Questions

Our objective is a software development approach to develop high-complexity software with high quality. The focus here is not to reduce the number of software defects and, consequently, the cost involved. Truly goal is to improve the programming experience and life quality of the software developers. Simultaneously high quality is built immediately into the software because software inspection is too late for the highest quality possible. Moreover, improving the quality of the software delivery to the users benefits society beyond having care for the health of the professional who programs the software certainly a fair procedure to do and a goal to pursue.

Therefore, it carried out a Grounded Theory about Mob Programming to validate the benefits of this collaborative approach in the Software Development of complex tasks and remote work. We sincerely believe in the potential of this new technique with an excellent opportunity to improve the software quality, help to manage the complexity and increase the satisfaction of the programmers.

Related to Research Questions, we follow the suggestion from Glaser (1978), who recommends effectively studying and uncovering the main problems of the participants. Additionally, the Grounded Theory research method strongly recommends refraining from formulating a research problem or a question upfront. The rationale behind this recommendation is:

- The GT method is meant to generate new theories, and having a preconceived research problem can cause the researcher to be limited in their explorations;
- The research problem should be the problem of the participants under study and should not be preconceived or forced, instead of it should be allowed to emerge.

The researcher obeys the caution of formulating a research question up front, then only choosing a general area of interest. The plethora of subject areas within Mob Programming makes choosing one a daunting task.

This research started by exploring Mob Programming on complex affairs as an area of scientific study. However, the pandemic of COVID-19 unexpectedly brought many challenges and opportunities, e.g., remote work was mandatory due to sanitary problems. Thus, the area of interest of this research gets wider, including Mob Programing to write high-complexity codes and remote work.

By Grounded Theory, the following research questions emerged during the empirical observations and were not formulated at the beginning of this study. In the write-up phase of the Grounded Theory, we decided to put these research questions to guide our readers.

Perhaps, in the future could be possible to make generalizations about other contexts and software development practices related. However, for now, we pursued that goal related to Mob Programming and revealed a set of unanswered questions about high-complexity tasks and remote work. These questions are about if Pair Programming, two developers, evolved into Mob Programming, composed of three or more developers, which could be profitable specifically in a complex task, along with verifying whether Mob Programming works well in a distributed way and observing its benefits.

Following are the research questions about Mob Programming in complex tasks and remote work, remembering that we used the Grounded Theory. Thus, in the last phase of the Grounded Theory, the writing phase, we decided to describe the research questions to make this text easy to read:

RQ1: **Is Mob Programming helpful when the task is complex?**

RQ2: **Is it possible to adopt Mob Programming to work remotely?**

1.3 Hypotheses

Although the research method used is Grounded Theory which will inductively generate a theory from raw data, looking to compare the findings in the quest to make a triangularization, beyond the use of multiple data sources, it also used the deductive research strategy through a questionnaire and source code analysis.

The initial phases of our Grounded Theory research method produced raw data about what we observed. Thus, it validated some aspects and provided new ideas. The classical version of Grounded Theory helps us to emerge one Theory to help software developers to find a better way to use Mob Programming based on a specific context related to complexity and remote work. Afterward, we verified the hypotheses to help those who are more familiar with the hypothetic-deductive research perspective:

1. The following are three hypotheses to answer the first unanswered question **RQ1**:

   H1: **Complex tasks are done with more quality using Mob Programming.**

   H2: **Mob Programming increases the learning and satisfaction of programmers.**

   H3: **If a team member needs to be absent in the Mob Programming session, the group maintains good performance.**

2. The following is the hypothesis to answer the second unanswered question **RQ2**:

   H4: **Remote Mob Programming has some of the benefits of face-to-face Mob Programming sessions when altogether team members are around the same computer when doing presential work.**

1.4 Research Strategy and Research Context

This thesis uses qualitative and quantitative data, and the research strategy is to observe companies, organizations, or institutions which adopted Mob Programming. In the quantitative observation part, preferentially open source projects or the source code produced using Mob Programming be open source, available on the internet, and the programming language used in the Mob Programming session described and audited by the author for this research.

Similarly, in the qualitative part, all questionnaires and transcriptions are available for audit and conforming to Brazilian law outside the internet environment. Moreover, all questionnaires and transcriptions will be available for audit for five years after the publication of the thesis. However, they are for the exclusive use of this research. In other words, it is not allowed by Brazilian laws to use the participants' data without their formal written approval for any other projects or purposes beyond this doctoral thesis.

The Grounded Theory performed here uses multiple data sources, objecting to generate a theory from the data collected. This Theory is associated with the research context. Thus, it is central to describe the context in which this study realized the data collection.

The majority of projects developed by the participants of this study were open-source projects to make them easy to audit. However, neither all were open-source, and a minor part was proprietary software.

Following is a brief description of the projects, institutes, organizations, and companies, toward contextualization of participants in this research:

University of Sao Paulo: is a Brazilian public university, the oldest University in Brazil, and the most prestigious higher education and research institution in Latin America.

Since Agile Methods formalization, software engineering education has also been impacted. Universities had to adapt their courses to suit these new software processes. In the Computer Science Department of the University of Sao Paulo, there is a discipline called eXtreme Programming Laboratory for undergraduate and Advanced Laboratory of Agile Methods and Software development for graduated students. The course is an opportunity to learn agile software development methods on real projects. It is a course offered annually by the Institute of Mathematics and Statistics of the University of São Paulo since 2001. During the classes, the students develop real projects with real customers.

The course has two names because it is available to graduate and undergraduate students, eXtreme Programming Laboratory for undergraduate students and Advanced Laboratory of Agile Methods and Software development for graduated students. The first name refers to eXtreme Programming, and the discipline aims at teaching agile methods in practice, considering several elements crucial for providing the student with knowledge and experience with agile software development. This discipline has provided extensive studies involving students, instructors, mentors, customers, professionals, and companies.

The discipline provides an agile environment similar to that found in industry and a proper infrastructure for developing agile projects. The presence of diverse roles in addition to instructors, including mentors (experts in agile methods or former students who have already been through an edition of the course that, along with the instructor, provides agile mentoring to all teams), and coaches (experienced students on software development or students that already know agile methods), is a newsworthy aspect. Students as developers interact with actual customers and spend time devoted to development work for real projects. Additionally, intra-team, inter-team, and whole-class learning techniques ensure a broad and continuous learning experience for students.

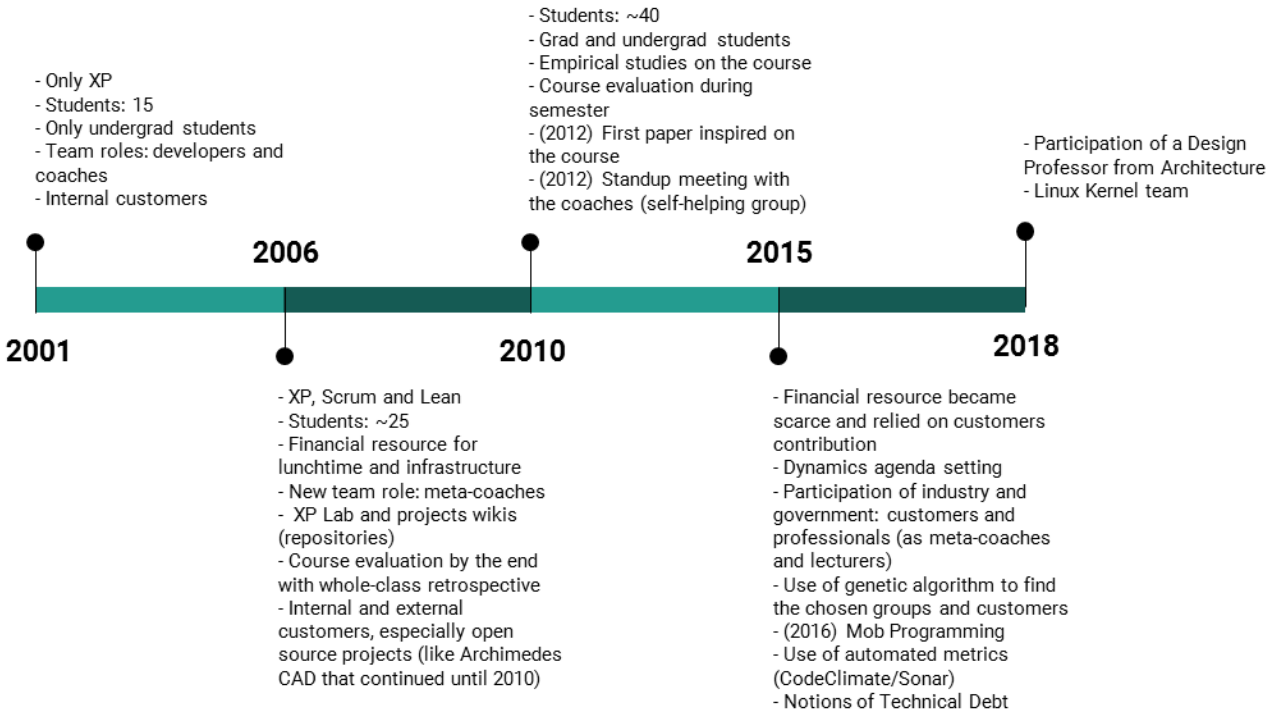


Figure 1.1: *Timeline of the experiences of more than 18 years of the existence of the eXtreme Programming Laboratory course.*

The projects are diverse Goldman *et al.* (2019). Some are brand new others involve legacy code. So, the development challenges are different for each team. In 2018, we had an exciting novelty, a project dedicated to the Linux Kernel. It worked fine. Now we have a group of students dedicated to it. Other projects were game creation Kattan (2019b), a food delivery app, a project to help persons with disabilities, and a web page to show how our representatives vote (radarparlamentar.polignu.org). In previous years we also had great diversity, from CAD for architecture to collecting source code metrics Kattan *et al.* (2018a).

Figure 1.1 is a timeline of the experiences of more than 18 years of the existence of the course, and for more details exists an experience report describing how the discipline evolved and improved based on student feedback Goldman and Santos (2019).

Recently, the course faced a new challenge: to teach the adoption of Agile Methods to develop projects that deal with low-level programming. In 2018, the discipline had a project related to the Linux Kernel IIO Staging Drivers. In 2019, the course had another project that aimed to refactor the new device's Ethernet driver, and it adapted the mindset to do Agile in these particular software development environments de Oliveira Rosa and Goldman (2020).

ccsl.ime.usp.br/wiki/Mezuro_2016: is a free/libre web platform for collaborative source code evaluation. Able to evaluate source code with the most popular SCMs (like Git and SVN), just by providing its URL. For now, it can evaluate C, C++, Java, Ruby, Python, and PHP source codes, but the Mezuro team are looking forward to supporting more languages in the future. Mezuro is continuously under development. To learn about this project, more information available in the Wiki (2018). Also is available a stable implementation of the Mezuro project and it is possible to create repositories or monitor the metrics of the existing ones at Mezuro.org (2018). The source code is at GitLab (2018)
Mezuro project aims provide a platform to compare projects and metric techniques, teach how to use metrics through configurations, analyze code, avoid technical debts, and disseminate code metrics usage and understanding. There's packages available for Debian Jessie and CentOS 7. Mezuro is composed of by three Ruby on Rails applications:

1. Prezento (2018) is the responsible for providing a web interface for users as well as taking care of authentication

2. KalibroProcessor (2018) does the heavy lifting of invoking the different metric softwares, parsing the results, aggregating through the source tree modules and interpreting them accordingly to each configuration. As you can see this is a lot of responsibilities for a single software, that's why we came up with an abstraction layer called Kolekti:

    - Kolekti (2018) provides a basic interface understood by KalibroProcessor so each metrics software integration can be developed separately in a way that KalibroProcessor has minimal knowledge about each software. Kolekti is a code metric results parser framework, and have several implementations for Kolekti:
        (a) KolektiAnalizo (2018) is the Generic parser for Analizo static code metrics collector, an implementation for Analizo (2018), a Java / C / C++ metric extractor
        (b) KolektiMetricFu (2018) is the implementation for MetricFu (2018), a Ruby metric extractor
        (c) KolektiRadon (2018) is the implementation for Radon (2018), a Python metric extractor
        (d) KolektiCCPHPMD (2018) is the implementation for PHPMD (2018), a PHP metric extractor

3. KalibroConfigurations (2018) is designed for the management of the metrics that will be gathered for each repository. It is a web service for source code analysis configuration, also be part of the Mezuro project. It is free software under the AGPLv3 license.

Rocket.Chat: is the leading free and open source team communication platform. Founded in 2015 by Engel (2015), Rocket.Chat began life as a Live Chat support tool built on Meteor. It has come a long way since then, received funding from US-based New Enterprise Associates in 2016 worth $5,000,000 USD and won multiple prizes such as a 2016 Bossie Award for Best Open Source Application and first prize in the 2017 edition of All Things Open's Startup Competition. Today, Rocket.Chat is an open source team collaboration platform that enables banks, NGOs, startups, and governmental organizations to have their own chat tool, customize its look and feel, choose their users, and securely manage data.

Users can set up Rocket.Chat on the cloud or by hosting their own servers on-premises and can be forked on Github and become a new platform altogether, thanks to its code being open source and thus openly available to the community.

Key features include free audio and video conferencing, guest access, screen and file sharing, LiveChat, LDAP[2] Group Synchronization, two-factor authentication[3] (2FA), End-to-end encryption (E2EE), Single Sign-On[4] (SSO), dozens of OAuth[5] providers and unlimited users, guests, channels, messages, searches, and files.

Customization is enabled with the option of adding or removing features and selecting your own integrations, plugins, themes, mobile and desktop apps.

Rocket.Chat is a fast-growing platform that is now installed on over one hundred and eigthy thousand servers and counts over ten million users worldwide as well as having an active, passionate community of over 700 developer-contributors who help core team of developers from Rocket.Chat to constantly improve the product.

The long-term vision of Rocket.Chat is to replace email with a real-time federated communications platform and to establish a marketplace that will nurture app-building and offer services to enable businesses to be built using Rocket.Chat.

Cucumber.io: is an open source project, and now also a company. But what kind of a company? There is a post from Wynne (2016) explaining it, loosely based around a talk Tooke and Wynne (2016) gave at QCon London, about the hard work, the decisions, and the fun into trying to build a sustainable business around this popular project.

About the business plan, Aslak Hellesøy likes spreadsheets, reckons there were some millions of downloads of Cucumber last year, double the figure from two years before. As Cucumber base of open-source users grows, means more demand for the Team give training and consulting, and by delivering on that demand Cucumber get revenue, which then makes more time for working on the open source. Then, because Cucumber team makes the Cucumber project better, it gets even more popular, in a virtuous cycle.

As they spent time with their users in consulting practice, they noticed there was a need for something else, something a bit more polished than the open source tool. When they are trying to adopt Behavior Driven Development[6] (BDD) in an organization where there is some resistance, they saw an opportunity for a product.

Cucumber team decided to try Mob Programming. They are all eXtreme Programmers. So nobody needs to convince them about the benefits of Pair Programming. In practice, however, a

[2]The Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.

[3]Two Factor Authentication (2FA) is an extra layer of security that requires not only a password and username but also something that the user has on them.

[4]Single sign-on (SSO) is an authentication process that allows a user to access multiple applications with one set of login credentials. SSO is a common procedure in enterprises, where a client accesses multiple resources connected to a local area network (LAN).

[5]Open Authorization (OAuth) is an open standard for token-based authentication and authorization on the Internet.

[6]Behaviour Driven Development (BDD) is a synthesis and refinement of practices stemming from Test Driven Development (TDD) and Acceptance Test Driven Development (ATDD). BDD is also referred to as Specification by Example.

lot of the old codebase had ended up getting built solo, people in hotel rooms and on planes and so on. So they did not have that shared consciousness about the code. They wanted to fix that too.

Mobbing is like turning Pair Programming up to eleven people and all of the good stuff you get about learning from each other, bringing the quality of the code up to the standard where everybody is happy with, these aspects are massively accentuated by mobbing.

Another eXtreme Programming (XP) practice, the collective code ownership gets turned up to eleven because everyone owns every line of code written. Everyone has a chance to say something about every line of code written, so all are concerned about the source code. Everybody gets a chance to be heard and can resolve those issues either by helping them understand why they do not need to worry or by actually factoring that concern in with a change.

Cucumber is a remote team. Matt Wynne lives up in the wilds of the Scottish Highlands. Steve is in Southampton. Aslak is in London. Julien and Romain are in Wavre, just outside Brussels. It is really hard to work on the same computer in the same space at the same time.

However, they have taken the decision that they want to be able to live where they want to. There are benefits to being able to be in the own space, being comfortable in your surroundings, being near your family.

There is something special about being in a work environment where at the same time also is propitious living family moments. Although, the challenges that come with being remote mean the need to find new tools to enable mobbing to work.

They use Slack for team chat and Screenhero to enable several people to share the same computer at the same time. Thus, a person could work with the tools that are comfortable in its use. Usually, they have a second screen, either a phone or a tablet or just a second monitor with a Google Hangout to enable they are all looking at each other, talking to each other using Google Hangouts. And that is how they code every day.

Something that amazes the CEO about the Mob is how resilient it is. On a small scale, suppose that a person has to take a phone call and disappear for 15 minutes, so come back. The Mob Programming session is rolling on. The team had been writing high-quality code, the whole time. Another seemingly positive aspect of Mob Programming is the fact of to be easy to catch what is going on at that moment and start to contribute within a matter of minutes.

Equally, on the bigger scale, if a person has to go away for a few days of training, maybe this person will be in a different time zone, the mob carries on. The Mob team are swapping different people out each week to go and deliver the training courses, but the consciousness is still there, and the programming task is moving forward all the time.

In the CEO crazy moments, imagines building a team where there were enough people who live in enough different time zones to the Mob Programming session keep going unstoppable twenty-four hours a day.

Running and growing Cucumber business is an adventure. The CEO is a business owner and a developer at the same time. Mike Feathers has been writing recently about the Symbiosis between code and organization, and this is something they live and breathe every day. The majority of the development team are also major shareholders in the business.

Because they get to build our product in the way that we know works well, in the end, that feels like play. Fun alone is not an objective, but Matt Wynne thinks it is possible to find a kind of joy in almost anything, even writing a sales proposal. What they really love to do is to learn and improve themselves, thus, developing their mastery. The CEO thinks Mob Programming feels playful for them. If you have done Pair Programming and worked in XP team, you will know how playful that can feel. When you are playful, you are relaxed. When you re relaxed, you make your best decisions.

Hashload: HashLoad Community is an open-source community hosted on GitHub. They develop some frameworks for Delphi. An example is Boss, an open-source dependency manager inspired by npm for projects developed in Delphi and Lazarus. Another HashLoad open-source project is Horse an Express-inspired web framework for Delphi and Lazarus. Designed to ease things up for fast development in a minimalist way and with high performance.

SindicoNet: Platform leading in Brazil for content, technology, and reference services for trustees, condominiums, and condominium administrators.

SindicoNet is the largest communication and services portal for trustees and condominium administrators in Brazil and has a great purpose of improving the lives of millions of Brazilians who live in condominiums.

The platform, which has more than 1 million visits per month, is also a reference in training liquidators and is responsible for the largest and most significant content and business event in the sector, the SíndicoNet Connection.

SíndicoNet is a portal 100% focused on life in condominiums and covers everything a property manager, manager, or condominium resident needs to know to live in harmony: whether in commercial or residential development. The team's goal is to offer quality content and first-class service to all users of the portal: whether residents who want to ask a question on our channel, asking questions, to entrepreneurs in the sector who register their companies to offer their services to property managers in your city.

Cotei Bem: CoteiBem is the SindicoNet platform focused on streamlining and simplifying condominium quotes.

Through it, suppliers expose their brand, attract customers, and trustees receive quotes from more than 5,000 qualified service providers evaluated by thousands of managers throughout Brazil. The quotation process becomes more transparent, agile, and economical.

A product developed by those who really understand the subject. Since 1996, SindicoNet has provided a network of information, tools, and services that elevate management and transform the experience of living in condominiums.

Shift: The shift is a Brazilian information technology company for diagnostic medicine. The combination of knowledge in technology with the concern to promote positive and lasting relationships explains the consolidation of Shift as one of the most renowned brands in Brazil and its expansion to Latin America.

Since 1992, Shift has been developing technological innovations to provide more efficiency, agility, and competitiveness for the laboratory segment in Brazil and Latin America.

The shift is responsible for a significant share of the volume of tests carried out in Brazil, where quality management is directly linked to our business. Around 80 percent of Shift's customers have at least one of the main quality certifications in the country, which demonstrates the relevance and demanding profile of the group of laboratories that rely on our solutions.

In addition to operating in 23 Brazilian states, Shift, which processes more than 40 million tests per year, is also present in Argentina, Paraguay, Bolivia, and Uruguay, where it offers solutions for the main laboratories in South American countries.

The shift is a knowledge company, and its strategic guidelines are the technological vanguard and entrepreneurial intelligence, which guide the vision of being a reference in the health sector in Brazil.

Multidisciplinary teams composed by business analysts, developers, biomedical professionals, and consultants create and manage the information system, integrations, and equipment automation, which provide security and intelligence to the processes from the beginning to the end of the laboratory operation.

The Brazilian Aeronautics Computing Center of Sao Jose dos Campos (CCA-SJ): In Do Brasil (1988) CHAPTER II, "The Armed Forces" of the CONSTITUTION OF THE FEDERATIVE REPUBLIC OF BRAZIL, Article 142. The Armed Forces, comprised of the Navy, the Army, and the Air Force, are permanent and regular national institutions organized based on hierarchy and discipline under the supreme authority of the President of the Republic and are intended for the defense of the Country, for the guarantee of the constitutional powers, and, on the initia-

tive of any of these of law and order. (Constitutional Amendments No. 18, 1998; Constitutional Amendments No. 41, 2003).

The italo-Brazilian binational project called AMX, designated in the Brazilian Air Force as A-1. It is evidence of the investment and ability of the Brazilian Air Force to create innovative technologies Figure 1.2. The AMX project has written a success story in the Italian Air Force by having flown in combat alone, in Afghanistan, Libya, and Kuwait, for more than 18,000 hours.



Figure 1.2: *Author Fab_oficial. A-1M at Campo Grande Air Base, Brazil, during EXCON Tapio 2022. Instagram, August 30 of 2022.*

According to de Sá *et al.* (2019a) and Brasil (1980), the military hierachy is the order to authority, at differente levels, within the structure of the Armed Forces. The ordination is done by posts or graduations; within the same post or graduation is done by seniority at the post or graduation. Respect to the hierarchy is embodied in the spirit of compliance with the sequence of autorithy.

Within the scope of the Brazilian Air Force, the Aeronautics Computing Centers have the purpose of managing the Information Technologies Systems and Services under their responsibilities DECEA (2011).

The adoption of agile methods at the Brazilian Aeronautics Computing Center of São Jose dos Campos was an initiative of its systems developers. Scrum was chosen as the development framework de Sá *et al.* (2019b).

The Brazilian Aeronautics Computing Center of Sao Jose dos Campos (CCA-SJ) is an organization of the Brazilian Air Force (FAB). Its mission is to develop and support software systems and flight simulators for the FAB.

In the Figure 1.3 is a GRIPEN replica in exposition at a Brazilian Shopping Center. The Gripen Design and Development Network objective is to support, in the engineering areas, tests and tests, integration, and modernizations, in addition to working on the development of Gripen evolution software worldwide. Such actions aim to expand Gripen's versatility in order to make it respond in the best way to the interests of the Brazilian Air Force.

As a military organization, the CCA-SJ follows a rigid command hierarchy, where software developers, computer engineers, system analysts, and all staff are military personnel. A few years ago,



Figure 1.3: *Author AeronauticaOficial. GRIPEN replica in exposition at BarraShoppingSul, in the city of Porto Alegre in Brazil. Facebook, September 16 of 2022.*

the development teams adopted Scrum to develop software and flight simulators. In the beginning, some problems have arrived. Two years later, the adoption of Scrum at the CCA-SJ became a success story de Sá *et al.* (2022).

Who knows a little about Scrum probably knows that one of its creators, Jeff Sutherland, was a fighter pilot for the United States Air Force (USAF) during the Vietnam War. Sutherland created Scrum years after the war, but several of his military experiences as a combat pilot contributed to this successful idea Sutherland and Sutherland (2014).

Both Scrum and other Agile methods are focused on individuals and their interaction. Its basis is an environment where people are encouraged to work collaboratively to achieve a common goal that delivers value to the customer. Scrum teams are self-organized, meaning no one tells them how to perform their job. Besides, they are multifunctional, holding all the necessary skills to complete their work. The Scrum team model improves flexibility, creativity, and productivity.

Notwithstanding the apparent contradiction between agile teams with horizontal interaction between their members and organizations based on command and control structures, it is clear that compatibility in the application of agile methods in these strongly hierarchical organizations is possible as very well demonstrated in de Sá (2022) and de Sá (2023).

The CCA-SJ organization started with Scrum adoption within development teams that became an Agile transformation. Culture and mindsets are changing. Finally, while the CCA-SJ is reaching new progress with Scrum and Agile, other organizations inside the Brazilian Air Force are already in contact with the CCA-SJ organization to share experiences and learnings. The main goal in this knowledge domain for the CCA-SJ is to become a reference in Agile methods within the Brazilian Air Force.

The modern Brazilian Air Force is continuing to improve its software development processes.

Also, it has a long history of international collaborations in fighter aircraft development, including simulators, embedded systems, etc.

The figure 1.4 occurs on November 3rd and 4th, a delegation from the 1st Air Defense Group and technicians from the Logistics Group of the Anapolis Air Base (BAAN) visited the facilities of the Gripen Design and Development Network and the Gripen Flight Test Center, at EMBRAER, in Gaviao Peixoto, Sao Paulo, Brazil. The main objective was the exchange of knowledge between SAAB technicians and future pilots, maintainers, and systems operators of the new fighter vector of the Brazilian Air Force, in addition to acting in the development of Gripen evolution software worldwide.



Figure 1.4: *Author Anapolis Air Base (BAAN). A delegation from the 1st Air Defense Group and technicians from the Logistic Group at Anapolis Air Base (BAAN) visited the Gripen Design and Development Network and Gripen Flight Test Center to exchange knowledge between SAAB technicians and future pilots, maintainers, and system operators of the new fighter jet of the Brazilian Air Force, in addition to acting in the development of Gripen evolution software worldwide, in the city of Anapolis in Brazil. Fab.mil.br, Edition: Air Force Agency - Review: Major Oliveira Lima. November 10 of 2021.*

Hunter Industries: Founded in 1981, Hunter Industries is a family-owned manufacturer of best-in-class solutions for the landscape irrigation, outdoor lighting, dispensing technology, and custom manufacturing sectors. Headed by CEO Greg Hunter, our Global Operations Team provides leadership for the entire company.

The core mission of Hunter Industries will always remain the same: to deliver valued products and services backed by unwavering customer support, grow the company conscientiously, and remain true to the culture that makes our employees proud to work at Hunter.

Hunter Industries offers thousands of products in over 120 countries, including a complete spectrum of water- and energy-efficient solutions for residential, commercial, and golf course irrigation systems. We also offer industry-leading landscape, architectural, and facade lighting solutions through our FX Luminaire and Lumascape brands.

Senninger Irrigation is a leading designer and manufacturer of premier irrigation solutions for agricultural, horticultural, and industrial applications. With over 50 years of experience in more than 50 countries worldwide, Senninger is one of the most trusted names in the agricultural irrigation

industry.

Hunter Custom Manufacturing provides innovative manufacturing, molding, and production services of the highest quality to build long-term relationships with valued partners from diverse industries. Our global reach enables us to produce raw materials with more than 100 injection molding machines and deliver finished goods around the world.

As leading innovators of the away-from-home dispensing category, Dispensing Dynamics is proud to continue its longstanding tradition of developing washroom dispensing systems that offer unsurpassed reliability, design, performance, hygiene, sustainability, and value.

Mob Mentality: In an altruist initiative, Mob Mentality is a podcast hosted by passionate people interviewing other passionate people about Mob Programming. Founded by Chris Lucian and Austin Chadwick, they are two exceptionally clever and talented professionals. They are pioneers of the Mob Programming idea. On the Mob Mentality Show, Chris Lucian and Austin Chadwick discuss all things #agile and product development from a #MobProgramming perspective. In the spirit of "Just Sharing" (https://justsharing.dev/), they and special guests share their experiences toward inspiring people about experimentation and continuous improvement. As shown here (https://www.mobmentalityshow.com/), they are on YouTube, Apple Podcasts, Spotify, Google Podcasts, Amazon Music, Podbean, Twitter, LinkedIn, and more.

1.5 Full and Short Papers, Workshop, Doctoral Symposiums, Video, and Posters

Some publications were produced based on this research as a result of the participation at National and International Conferences; these publications have different formats, such as Full Papers, Short Papers, Workshops, Doctoral Symposiums, Videos, and Posters. They are following listed:

1. **Full Paper**: Kattan (2016b). Illuminated Arrow: a research method to software engineering based on action research, a systematic review and grounded theory. In 13th International Conference on Information Systems and Technology Management. CONTECSI USP, ISSN 2448-1041, Brazil, June 2016, pages 1971-1978. Available at: http://contecsi.fea.usp.br/index.php/contecsi/13CONTECSI/paper/view/3926/2570 (Accessed: July 2018).

2. **Poster**: Kattan (2017a). Those who fail to learn from history are doomed to repeat it. In Agile Processes in Software Engineering and Extreme Programming: poster presented at the 18th International Conference on Agile Software Development, XP 2017, held in Cologne, Germany, on May 22-26. URL https://doi.org/10.13140/RG.2.2.20864.02563

3. **Doctoral Symposium**: Kattan and Goldman (2017). Software Development Practices Patterns. In Agile Processes in Software Engineering and Extreme Programming, Hubert Baumeister, Horst Lichter, and Matthias Riebisch (Eds.). Springer International Publishing, Cham, Pages 298–303. URL https://doi.org/10.1007/978-3-319-57633-6_23

4. **Video**: Kattan (2017b). Pares de policiais para segurança, pares de programadores para qualidade. Presentation at TDC 2017 Sao Paulo. Available at: https://thedevconf.com/tdconline (Accessed: July 2018).

5. **Full Paper**: Kattan *et al.* (2018a). Mob Programming: The State of the Art and Three Case Studies of Open Source Software. In Agile Methods, Viviane Almeida dos Santos, Gustavo Henrique Lima Pinto, and Adolfo Gustavo Serra Seca Neto (Eds.). Springer International Publishing, Cham, Pages 146–160. URL https://doi.org/10.1007/978-3-319-73673-0_12.

6. **Poster**: Kattan (2018b). Swarm or Pair? strengths and weaknesses of Pair Programming and Mob Programming. In Agile Processes in Software Engineering and Extreme Programming: Poster presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25. URL https://doi.org/10.13140/RG.2.2.18105.06249.

7. **Workshop**: Kattan (2018c). To Swarm or Not To Swarm (Programming and Review Simultaneous in Pairs). In Agile Processes in Software Engineering and Extreme Programming: WorkShop presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25. URL https://xp2018.sched.com/event/DxzC (Accessed: July 2018).

8. **Doctoral Symposium**: Kattan (2018a). Theory of Altruism on Software Development Practices Patterns. In Agile Processes in Software Engineering and Extreme Programming: presented in the doctoral symposium at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25. https://xp2018.sched.com/event/E0LA (Accessed: July 2018). Association for Computing Machinery. ISBN 9781450364225. doi: 10.1145/3234152.3314991. URL https://doi.org/10.1145/3234152.3314991.

9. **Short Paper**: Kattan *et al.* (2018b). Swarm or Pair? strengths and weaknesses of Pair Programming and Mob Programming. In Agile Processes in Software Engineering and Extreme Programming: Short Paper presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25. doi: 10.1145/3234152.3234169. URL https://doi.org/10.1145/3234152.3234169.

10. **Full Paper**: Kattan (2019b). Kattan H.M. (2019) Mob Programming and Simultaneous Style Pair Programming in the Development of a Battle Royale Game: An Action Research. In: Meirelles P., Nelson M., Rocha C. (eds) Agile Methods. WBMA 2019. Communications in Computer and Information Science, vol 1106. Springer, Cham. URL https://doi.org/10.1007/978-3-030-36701-5_4.

11. **Full Paper**: Kattan (2019a). Herez Moise Kattan. 2019. Software Development Practices Patterns: from Pair to Mob Programming. Proceedings of the Regional School of Software Engineering (ERES). In Anais da III Escola Regional de Engenharia de Software, outubro 22, 2019, Rio do Sul, Brasil. Brazilian Computing Society - SBC, Porto Alegre, Brasil, 147-156. Available at: https://sol.sbc.org.br/index.php/eres/article/view/8507.

1.6 Claimed Contributions

This thesis presents a Grounded Theory of Mob Programming for writing high-complexity codes and remote work. The grounded theory is the main contribution and is based on a research study involving 200 software developers across different organizations from Brazil and USA. The resulting

theory of Mob Programming for complexity and remote work explains how software development teams take on the informal and implicit ways of organizing the programmers' work in a Mob Programming session. Room setup, driver rotation techniques, retrospectives, and structure breaks are some practices involved in a Mob Programming session correlated with the team roles observed as more stable in the teams.

The initial evidence for the effectiveness of Mob Programming by its proponents and early adopters is quite encouraging. It still requires validation through rigorously designed empirical research, so more organizations could adopt it fully understanding the conditions that accentuate its benefits and minimize potential risks Balijepally *et al.* (2017).

This research collaborates with software practitioners in this endeavor, helps establish Mob Programming as a viable agile practice and speeds up its adoption, and discovers if the approach is advantageous for which context.

The claimed contributions of this thesis are related to both: academics and the industry of software. Some of them are closer to Universities, academics, and researchers. On the other hand, there are other contributions related to the Market, IT-related workers, and practitioners who work to produce software.

This thesis also presents a discussion of the research results in light of existing literature, which further supports the team roles, Mob Programming practices, Mob Programmers' behaviors, Mob Programming communication space, challenges faced when start adopting Mob Programming, Mob Programming to manage the complexity as well to write high-complexity codes and aspects related to remote work are also identified in this research.

We are not alone in studying software developers' behaviors, what software practitioners think and do when making value decisions, and articulating what specific values guide their daily work and how. Ferrario and Winter (2023) developed a relevant approach to the understanding and a framework for the effort of human values in the Software Engineering field.

A description of Grounded Theory as a research method, its application in this research with examples, and reflections on the challenges faced in using Grounded Theory and strategies for overcoming them are also presented.

Table 1.1 show a correlation between each of the claimed contributions with the closest correspondent, academic, the industry of software, or both of them.

Claimed Contributions	**Researchers**	**Practitioners**
CC1		X
CC2	X	X
CC3	X	X
CC4		X
CC5		X
CC6	X	X
CC7	X	X
CC8	X	X
CC9	X	X
CC10	X	X
CC11	X	X
CC12	X	X

Table 1.1: *Relation of the proximity of the Claimed Contributions with the Researchers in an academic environment or with software developers in the industry of software or with both of them. The letter "X" means the connection of the Claimed Contribution with Researchers, Practitioners, or both of them.*

Following the Grounded Theory using **Memos**, it was discovered the **Categories**, using the **Constant Comparison Method**, help us to think and reflect on them, which permitted the emergence of the two **Ontologies**. Afterward, using the 6C's model was suggested six **Theories**.

The related claimed main novel contributions of this thesis are following described:

CC1: **Increasing the satisfaction at work for the developers is helpful to improve the experience of programming and programmers' life quality.** Moreover, a team more motivated using Mob Programming produces a higher-quality software product. Although we do not get a conclusion about reducing group turnover, a satisfied professional probably not leaves the organization as an unsatisfied worker. The long-term effects of satisfaction and an understanding of the satisfaction of the programmers are vitally significant to top management. Because it could help in the reduction of turnover, as well as maintain the professionals talented in the organization.

CC2: **Mob Programming is an excellent way to organize the work of programmers when the task is complex.** Because beyond the complexity management, the software is developed with more quality using Mob Programming. Moreover, other important aspects, e.g., the collective ownership of the code done with more effectiveness.

CC3: **Remote Mob Programming has been observed preserving the benefits of face-to-face Mob Programming when the programmers sit together around the same computer during a Mob Programming session.** Also, an additional advantage observed in the dependency queue reduction is described in chapter 8 "Results about to Remote Work".

CC4: **Mob Programming is related to the quality of the software product.** It was observed that the source code produced by Mob Programming has a higher value. Moreover, it is described in chapter 7 "Findings Related to Managing the Complexity" this high-quality software helps to manage complexity and has a higher adherence to the codification standard. The quality has improved due to a better understanding of the problem domain, and because code reviews are done in real-time, it was observed that the code produced followed all standards and good programming practices.

CC5: **Deep knowledge about Mob Programming team roles, including Remote Mob Programming, is interesting because it contributes to increasing the diversity in software development teams making the software product reflect better the distinct views of the final users. It could help to fit each software developer's skills with the team roles.** There is a diversity kind of people in society, such as introverted, extroverted, women, men, and so on. A team reflecting the diversity of personalities of people that will use the software developed could maximize the chance of this software product reflecting better the user needs. And, it knowing the team roles could help in this match. Moreover, it observed well-defined team roles help all team members maintain the focus on problem-solving, as described in chapter 6 "Challenges to Start Adopting Mob Programming" section 6.2 "Keeping the Focus."

CC6: **Expand the duration of the progression of the work without a stop, continuous work in turns, through Remote Mob Programming.** A team where there are enough people who live in enough different time zones to the Mob Programming session keep going unstoppable twenty-four hours a day.

CC7: **Boost the Sharing of Knowledge because the new technologies and frameworks are fundamentals in software development.** Thus, software developers need to learn them as fast as they can. This thesis verified that Mob Programming is an efficient practice toward providing refined learning sessions while the developers work to produce software products while they are also simultaneously learning. Mob Programming increases the learning and satisfaction of programmers.

CC8: **Confirmation of the experience report of Freudenberg and Wynne (2018), about Mob programming to be surprisingly inclusive. If done right, it can provide a safe and nurturing environment for the fledgling programming skills of novice software developers or new team members. Not only this, Remote Mob Programming offers significantly more support and flexibility for family life or other interests.** The experience report Freudenberg and Wynne (2018) describes a case when an autist person enjoys using Remote Mob Programming, and it was surprisingly an inclusive practice in this way. It was observed that if the team had kind behavior when somebody else was vulnerable helped to create a safe and innovative Communication Space.

CC9: **If a team member needs to be absent in the Mob Programming session, the group maintains good performance.** In general, it does not cause an impact. However, it observed when more experient members are away from the Mob Programming session, the senior members missing originates an issue that causes a delay in tasks that can sometimes take a little longer to be performed.

CC10: **The two Ontologies are briefly described in Table 1.2.** Ontologies provide agents with the qualification to start communication in a well effective way and with unprecedented immediacy. Through an agreed upon ontology, a group of agents can build a tailor-made Communication Space or accept an off-the-shelf one and immediately start coordinated action or collaborative problem solving da Silva and Agusti-Cullell (2008). We proposed the following Ontologies:

Ontology	Description
Mob Programming Team Roles Ontology	Perhaps, the main contribution was starting to write the Mob Programming Team Roles Ontology because writing the Team Roles as an Ontology helps those who are beginning to use Mob Programming guiding them and who have more experience with Mob Programming could aid by the addition of new roles or contribute to this field of study more scientifically.
Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors	Writing an Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors was a great challenge. However, I hope to help software developers from every country could have a better, safer, and more innovative Communication Space to work.

Table 1.2: *This table contains only a short description of the two Ontologies since more details beyond this brief description of Mob Programming Team Roles Ontology are available in Chapter 4, and about the Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors in Chapter 5.*

CC11: **Discovery of six Theories related to Mob programming. Following are the first four Theories, with their brief description in Table 1.3.** Each short description has a link to the respective section that contains the full description of the Theory using 6C's model. The number of discovered theories is six, and due fit the page, it was split into two tables to make it easy to read.

Theory	Description
Well-Defined Team Roles Theory	When the team uses mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Was observed a correlation with the challenge of keeping the whole Mob Programmers focused on solving the problem.
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable	Structure Breaks are fundamental and fruitful in managing complexity and preventing Mental Exhaustion. Five different contingencies were identified: Learning something new together in a group or solo; Mini-Retrospectives; Pair Programming; Members Rotation among the Teams; Pair Programming Simultaneous Style.
Embracing Vulnerabilities with Kindness Theory	We identified the positive effect of having a Kind and Fearlessness Programmer because this behavior contributes toward a Safe and Innovative Communication Space.
Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming	We identified the positive effect of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists that contribute to outcomes or goods of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists, these benefits are: closer team; knowledge sharing; manage complexity; higher quality software; adherent to the code standard; clean code and safer; reducing the chance of a damaging cyber incident; maximizing resilience.

Table 1.3: *This table contains only a short description of the first four Theories since more details beyond this brief description of Well-Defined Team Roles Theory are available in Section 4.10, and about the Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable in Section 5.17, and about Embracing Vulnerabilities with Kindness Theory, in Section 6.10, and about the Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming in Section 7.15.*

CC12: **Following are the last two Theories, with their brief description in Table 1.4.** The total number of discovered theories is six, and due fit the page, it was split into two tables.

Theory	Description
A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage	Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors
Theory of Benefits of Remote Mob Programming	Outcomes or effects of having Remote Mob Programming are that the Mob Programming benefits are preserved; the possibility of working from anywhere; family and pets closer; little-to-no-commute; greater inclusivity; non-stop working in different time zones.

Table 1.4: *This table contains only a short description of the last two Theories since more details beyond this brief description about a theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage are available in Section 8.12, and about the Theory of Benefits of Remote Mob Programming are available in Section 8.13.*

1.7 Thesis Structure

The thesis structure consists of three parts. The first is theoretical. The second is empirical. The third is for the documents related to this thesis:

1. **The first part consists of three chapters. They are about the insertion of the reader into the theme. Also, it accommodates the introduction, contextualization, literature review, and research method description.**

    - The chapter 1 "Introduction" is for the contextualization, participant descriptions, list of the claimed contributions, the objective, the introduction of the research strategy, and thesis structure.
    - The chapter 2 "Literature Review" started with a Minor Literature Review with some basic terminology used by peers. As the Theory began to emerge, the researcher conducted an extensive literature review to see how the literature in the field related to their emerging Theory. Mob Programming is a natural evolution of pair programming, and Mob Programming is newer than pair programming. Mob Programming is a new approach with only experience reports without an acceptable research method. Mob Programming has similar aspects related to Pair Programming. Pair Programming has extensive literature associated with it and uses more appropriate research methods rather than Mob Programming. Thus, pair programming was studied much more than Mob Programming. Literature Review beyond Mob Programming included the literature on Pair

Programming because of their similarities, and some researchers studied both together. Moreover, it provides a broad view of the related studies.

Further, the Literature Review also includes Complexity, Information Flow, Knowledge Sharing, and Ontologies because all these concepts were used in this thesis.

- The chapter 3 "Research Design" discussed the research approaches and strategies, strengths, and weaknesses in the Grounded Theory. Limitations are discussed, as well as the reasons for the choice of grounded theory.

2. **The second part consists of four chapters to provide a detailed description of the Empirical part of this thesis, in other words, the grounded theory research method results.**

   - In the chapter 4 "Mob Programming team roles ontology" we presents the core of our Grounded Theory of Mob Programming Team Roles. By explicitly acknowledging this Ontology on Mob Programming Team Roles, the entire team could play to people's strengths and celebrate how their individuality can best contribute to the overall effectiveness and flow of the Mob Programming Session.
   - The chapter 5 "An Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors" presents the core of our grounded theory of An Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors. The theory explains how software development teams perform balancing acts on a set of integrated practices, in order to become an effective Mob Programming team. The integrated practices are Driver Rotation, Retrospectives, Structure breaks, Group Size, Room setup, and Automation. This chapter also describes the informal behaviours which exist in Mob Programming teams.

     By explicitly acknowledging this Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors, the whole team could suggest people's behaviors to better contribute to the overall effectiveness and flow of the Mob Programming Session and the Communication Space.

     We begin by presenting the notions of Communication Space and Productive Dialogue. The reason is they will be used in the subsequent sections and chapters. Then we describe the practices that contribute to a productive Mob Programming session. Afterward, we detail the behaviors of Mob Programmers observed as fruitful. We connected in this chapter what has been observed in practice with related literature that helps us with the explanation. Afterward, a discussion on the theory of planar graphs can be related in some way to the positioning of Mob Programmers. At the end of this chapter, we present the 6 C's model.
   - The chapter 6 "Challenges to Start Adopting Mob Programming" presents the core of our grounded theory of the Challenges faced for starting the adoption of Mob Programming. These categories are grounded in the data observed, questionnaires applied, and interviews done.
   - The chapter 7 "Findings Related to Managing the Complexity" presents the core of our grounded theory of the Mob Programming for writing code with high complexity. These categories are grounded in the data observed, questionnaires applied, and interviews done. It is described how Mob Programming could help to Managing the Complexity.
   - The chapter 8 "Results About Remote Work" presents the core of our Grounded Theory of the Remote Mob Programming. These categories are grounded in the data observed, questionnaires applied, and interviews done. It is described some differences between Mob Programming and Remote Mob Programming.
   - Finally, in the chapter 9 "Conclusion" summarized the Key Results.

3. **Brazil has a centralized registration process for ethics committees and requires institutional-level ethics committee approval for each trial site. The National Research Ethics Commission (CONEP) is the central body responsible for coordinating the network of institutional ethics committees, known as Committees of Ethics in Research (CEPs) in Brazil. CONEP is a collegiate advisory body directly linked to the National Health Council (CNS), a permanent body within the Unified Health System (SUS). The SUS is the public health system in Brazil and is part of the organizational structure of the Ministry of Health.**

   The third part consists of eight appendices. It is about several documents related to this thesis due to the Plataforma Brasil[7] requirements. Also, it permits the reader to audit, reproduce, verify or learn about the various documents needed in this research, including the ethical ones. Plataforma Brasil is the digital working platform of the CEP/CONEP System. It brings together researchers and institutions, CEPs, and CONEP itself, in a virtual environment that offers all the tools necessary for the Brazilian Government to evaluate and monitor the research involving human beings in Brazil.

   - Appendix A "Questionnaire in Portuguese" is the questionnaire in Portuguese reviewed and approved by the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP).
   - Appendix B "Questionnaire in English" is the questionnaire in English reviewed and approved by the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP).
   - Appendix C "Interview Script in Portuguese" is the Interview Script in Portuguese reviewed and approved by the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP).
   - Appendix D "Interview Script in English" is the Interview Script in English reviewed and approved by the *Plataforma Brasil* of the National Commission of Ethics in Research (CONEP).
   - Appendix E "TCLE" is the Informative Consent Document in Portuguese.
   - Appendix F "Informed Consent Document in English" is the Informative Consent Forms in English.
   - The approval of the Ethical Committee and documents related to Ethics are available in the Appendix G "Authorizations of the Ethics Committees".
   - Appendix H "Institution Official Consent Terms" are the Institutional Official Consent Terms of the companies, institutions, and organizations. Some participants are open-source projects without a formalization of their organization. This Appendix H is about the written approval of the formal ones.
   - Appendix I "Background of the Researcher" is about the background information of the author of this doctoral thesis, and it is needed due a result of the choice of this study to use the Grounded Theory research method and to accept the use of an interpretive approach because of the pursuit of generating a new theory. The role of the researcher is essential in how the phenomenon of Mob Programming is understood and explained. This section arranges background information of the researcher.

[7] Access to Plataforma Brasil is through the link: http://plataformabrasil.saude.gov.br

Chapter 2

Literature Review

Our Literature Review started with a Minor Literature Review with some basic terminology used by peers. As the Theory began to emerge, the researcher conducted an extensive literature review to see how the literature in the field related to their emerging Theory. Mob Programming is a natural evolution of pair programming, and Mob Programming is newer than pair programming. Mob Programming is a new approach with only experience reports without an acceptable research method. Mob Programming has similar aspects related to Pair Programming. Pair Programming has extensive literature associated with it and uses more appropriate research methods rather than Mob Programming. Thus, pair programming was studied much more than Mob Programming. Literature Review beyond Mob Programming included the literature on Pair Programming because of their similarities, and some researchers studied both together. Moreover, it provides a broad view of the related studies. Further, the Literature Review also includes Complexity, Information Flow, Knowledge Sharing, and Ontologies because all these concepts were used in this thesis.

This chapter describes the literature review of **Pair Programming** and **Mob Programming**. It also discusses the strengths and weaknesses of both approaches adopted. In the end, it is described the literature review on the **Complexity** of the software produced, **Information Flow, Knowledge Sharing**, and **Ontologies** because all these concepts were used in this thesis..

2.1 Introduction

The evolution of humankind developed social characteristics in the humans. Pair programming is a practice for software development teams, and it is also a process of informal review. It improves the code perceived quality and the technical level of the team. Pair programming boosts a collective feeling, both on ownership and responsibility for the system. Stimulating egoless programming idea Weinberg (1971) and Information Flow and Knowledge Sharing da Silva and Agusti-Cullell (2008), the software is owned by the team as a whole and individuals are not responsible for problems with the code; the entire team has a responsibility to address these problems.

I worked on the issue of productivity of pair programming without lost the benefits and published a possible solution in my Mastering thesis, the simultaneous style pair programming Kattan (2015), where is not need to sit side by side all the time to have the benefits of the traditional pair programming with an increase in the productivity of the software development.

There are related techniques, such as mob programming, where the whole team participates in around a computer described by Zuill (2014). Also, there is code review, another well know and applied alternative, which might increase productivity compared to pair programming according to Rajendran Swamidurai and Kannan (2014). So, this motivated us to do a more broad literature review in the context of these related techniques that allow cooperation or collaboration for software development.

2.2 Pair Programming

The source code is written by pairs of programmers who have distinct roles, sitting side by side and using the same computer. A partner is responsible for programming and thinks in algorithms and programming logic. While other observes the code produced, thinking more strategically about how to improve it and get the code simpler. Moreover, he also checks for possible errors and points of failure Begel and Nagappan (2008). These roles are respectively called pilot and co-pilot.

The idea is to develop software in pairs and not as an isolated task. This approach promotes up social aspects of programming, while one creates the other developer can do code review, discuss alternatives, check the quality of code and help in more complex parts Coplien and Schmidt (1995).

Some pair programming advantages beyond of the increase of quality perceived are:

1. Pair programming involves constant communication and collaboration, encouraging the dissemination of knowledge among the team members. Pair programming fosters confidence among developers Constantine (1995);

2. There is a decrease in the number of defects when using pair programming Nosek (1998);

3. Pair programming encourages companionship. It is a protective factor for conservation and decreases the dropout of women and novice programmers in computer science courses, thus, could be a low-cost approach for reducing the turn-over in software teams Williams and Kessler (2000);

4. According to Williams and Kessler (2003), the partner, who is not in possession of the keyboard, contributes with good ideas this helps maintaining discipline and concentration in productive activities Beck (1999).

Software products developed using pair programming generally get a higher quality compared with ones developed through solo programming. However, a literature review of pair programming revealed some drawbacks, such as a possible loss of productivity when using the technique, especially for experienced programmers and simple tasks. It was observed in the team if use pair programming by Nosek (1998), there might be an effort increase in the project.

Loss of productivity is a common criticism of pair programming since it devotes two professionals to perform the same job simultaneously on a single computer. There is a constant debate in academic and professional circles if the technical advantages justify this possible loss of productivity Allen Parrish and Hale (2004).

2.2.1 Concept View

Although Pair Programming is known as a technique associated with the eXtreme Programming, the creation of pair programming precedes the XP. There are many old tech-savvy reports, such as a report of Brooks (1975). It describes a situation when programming in college between 1953 and 1956, he and his pair wrote 1500 lines of code, which they executed without any error on the first attempt.

A pattern called Development in Pairs published in 1995, on the authors' views, some people occasionally only feel able to solve a problem with help. There are problems seem to be greater than individuals Coplien and Schmidt (1995).

Pair programming helps to decrease the number of software defects Coplien and Schmidt (1995). It helps to improve software quality, but it also increases the effort of software development. On an experiment, the pair spent more work hours compared to two developers working alone Constantine (1995).

Coplien and Harrison (2004) defend the advantages of making pairing compatible designs work together. Again, in this way, they can produce more than the sum of the two individually. Constantine (1995) observed a reduction in the number of defects when did pair programming. Nosek (1998)

found that pair programming improves code and algorithm quality, but also noted the increased development effort, as it observed that the pair spent more working hours than if a single programmer worked alone.

Figure 2.1 shows the essential role of Pair Programming. It is directly related to almost all the other practices of the eXtreme Programming of Beck (2000). Pair Programming centrality is evidenced in the fact of if a withdrawal it happens, would directly affect all the related practices Menzies *et al.* (2003).

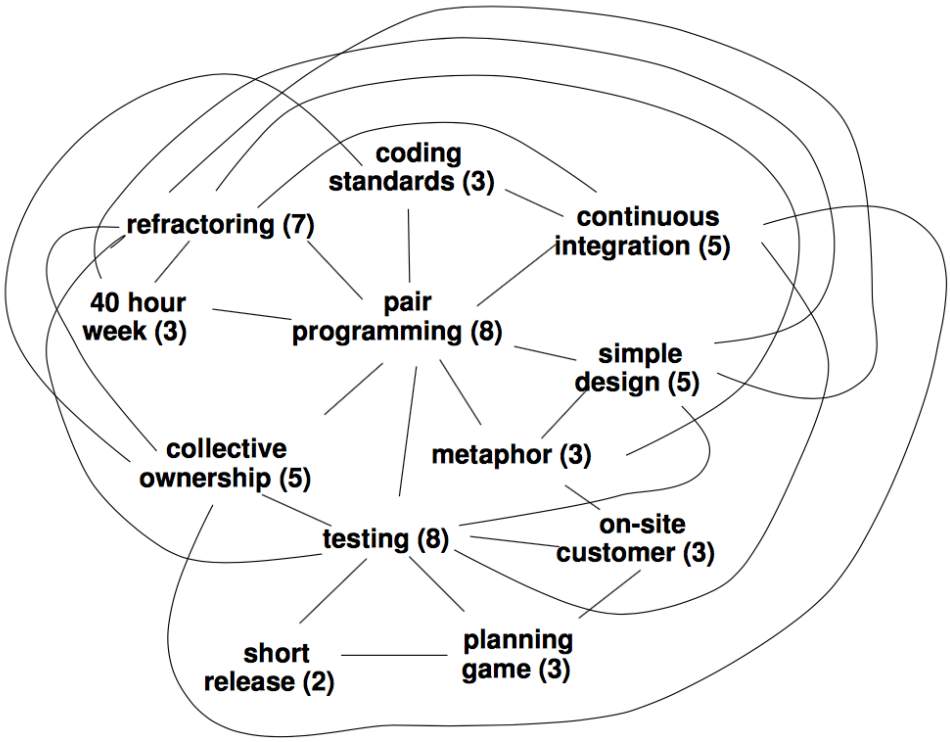


Figure 2.1: *An illustration of Menzies et al. (2003) about pair programming as a central practice. Observe in the diagram pair programming is directly related to almost all the other techniques of the eXtreme Programming from Beck (2000).*

2.2.2 Definition

A formal definition according to Begel and Nagappan (2008): **two** programmers working collaboratively at the same activity, sitting **side by side** in front of the **same computer**. While one person is writing the code, then another is paying attention to the work, looking for defects and suggesting improvements. There are similar approaches, using two monitors and mouses, but without any conceptual difference according to Al-Jarrah and Pontelli (2016).

2.2.3 Helpful to introduce newcomers and to reduce the time-to-benefit but increase the cost

A famous phrase by Brooks (1975), adding human resources to a late software project makes it later. Adding a new member to the team that does not know the system creates the problem of the first contact with the source code. The pair programming contributes to solving this problem because the new team member can pair with someone more experienced according to Williams and Kessler (2000) and Williams and Kessler (2003).

For Padberg and Muller (2003) and Padberg and Müller (2003), the code produced using the Pair Programming is more defect-free. They conclude, when market pressure is strong, to add

developers and to create more pairs, can accelerate the project and can increase its value for the business, but it also increases the cost.

2.2.4 Reducing the rate at which developers leave the team, programmer experience, and task complexity

Arisholm *et al.* (2007) and Parrish *et al.* (2004) carried out their work with experienced programmers. They found that there was a significant loss of productivity compared to two programmers working alone. There were some benefits to the quality, but they did not fully compensate for the loss of productivity of pair programming. However, according to them, the knowledge sharing that happens during pair programming is a very positive aspect as it reduces the overall risks of a project in the event any team member leaves and this may justify its use.

Hulkko and Abrahamsson (2005) have observed in four case studies the effectiveness of pair programming in disseminating knowledge. Hulkko and Abrahamsson (2005) and Vanhanen *et al.* (2007) also observed that it might be more useful in complex tasks.

The research of Dyba *et al.* (2007) reviewed fifteen papers on pair programming and all of them used programming tasks as a basis for the comparison done in their study. The number of individuals in the studies reviewed ranged from 12 to 295. When analyzing the result, it is possible to see the improvement in quality, reduction in the number of defects, but also, when using the pair programming, the increase of the effort in the project, when compared to the solo (individual) programming.

Table 2.1 describes a direct relationship between the advantages of pair programming with the programmer experience and task complexity. Only one study, though large and consistent, Arisholm *et al.* (2007) recognized the complexity of the task and the programmer experience.

Programmer' Experience	Complexity of the task	Is it advantageous to use pair programming?
Junior	Simple	Yes, provided that increased quality is the main goal.
Junior	Complex	Yes, provided that increased quality is the main goal.
Average	Simple	No.
Average	Complex	Yes, provided that increased quality is the main goal.
Senior	Simple	No.
Senior	Complex	No, unless you're sure that the task is too complex to be solved satisfactorily by an individual senior programmer.

Table 2.1: *Guideline for when to use pair programming of Dyba et al. (2007)*

2.2.5 Mechanisms to improve the performance of pair programming

Wray (2010b) published four mechanisms to improve the performance of pair programming: **chat**, **notice more details**, **fight against poor practices**, and **share and recognize competencies**. There was some feedback in the IEEE website that justifies this other publication Wray (2010a) with some of the comments about these following four mechanisms to improve the performance of pair programming:

1. The programmers' **chat** is a key point toward the better performance of pair programming, part of the effectiveness of pair programming is presumably due to an interaction between

programmers. When programmers chat about the problem in the moment of doubt, they find the solution faster and consequently productivity increases.

2. **More details** are perceived when doing pair programming. Thus, the pilot/driver frequently must be warned because probably not **noticed** the same issue, the reason is the human limitation of perception capacity that depends on the purpose of the search. There is a blind spot in the human attention on the unexpected, usually, have difficulty about notice something that is not paying attention. Fatigue decreases productivity, and changing the person from the pair typing on the keyboard during work can increase productivity. Pair programming reinforces the importance of keeping in mind that a beginner programmer increases her/his productivity when is pairing with someone more experienced.

3. The **fighting against poor programming practices** helps to increase the performance. Pair pressure helps to improve the programming quality. Pair Programming reinforces the importance following the programming standards and to program in the best possible way so that the code does not become bad and disorganized, this avoids rework and increases productivity.

4. **Sharing and recognizing competencies** cooperates with the dissemination of knowledge. Discover who are the experts on specific subjects and share this knowledge with the entire team, could help to increase the productivity. Wray (2010b) suggests the reworking of Dyba *et al.* (2007) toward it to consider its four mechanisms with new methods and criteria to effectively measure the performance of the team using Pair Programming.

An extension of the traditional pair programming approach called **Programming and review simultaneously in Pairs** of Kattan (2015), a kind of **Simultaneous Style Pair Programming**, benefit some productivity factors in software development compared to the traditional style Pair Programming.

Zieris and Prechelt (2021) report two elements of pair programming skill from good pair programming session: managing to maintain their Togetherness and keeping an eye on their session's Expediency.

2.2.6 Confidence, satisfaction, gender, and learning

According to McDowell *et al.* (2006) pair programming improves code quality, increases programmers' confidence and satisfaction (noted a 24% increase for women and 15% for men) and reduces student dropout in computer courses, McDowell *et al.* (2006) collected data from 554 students who attempted the course at the University of California-Santa Cruz. Pair programming stimulates programmers' motivation and satisfaction according to Muller and Padberg (2004).

The results reported by McChesney (2016) reinforce the view that pair programming contributes positively to student performance and confidence. Students report a positive experience, appreciating the overall benefits of collaborative problem solving while acknowledging there are trade-offs in terms of conflict resolution and time management. The study also suggests that repeated opportunities to practice pair programming enable the dynamics of pair programming to operate more smoothly.

Choi (2013) assess the relevance of gender in the context of pair programming, 128 students participated, being 93 men and 35 women. It observed no difference in the ability to program and solve problems, however, occurred a significant difference in the communication and compatibility of the pairs. The pairs of the same gender, in other words, man pairing with a man, and woman pairing with a woman have more compatibility. These same gender pairs could be used as a guideline when it comes to pair selection.

The Ruvalcaba *et al.* (2016) study showcases the strength of using a mixed methods approach to understand computer science learning. They approach draws from video analysis and case study approaches to gain a better understanding of how children from diverse backgrounds collaborate

while programming together on one computer. The quantitative data shows that White student pairs used a higher number of 'explanation' and 'suggested direction' observables interactions in an interval in their collaboration. They also found that the use of these "observable interaction in an interval" was low among the Latino student pairs. The Latino students may instead use skilled nonverbal communication to reach an agreement and make suggestions, thus being missed by our system of quantitative analysis. Across all student groups, regardless of cultural background, the two most common observables interactions in an interval were 'questions' and 'nonverbal looks'. The first most common "observable interaction in an interval", 'questions', suggests the importance of teachers encouraging students to ask questions during pair programming. Thus, they suggest it is important that makes students feel comfortable asking questions when learning with technology especially for underrepresented groups such as Women and Latinos who may have less experience or confidence with technology.

Missiroli *et al.* (2016) investigation shows that there is a definite advantage to early exposure of inexperienced programmers to Agile, not only as a project development process but also as a teaching tool. Pair programming is the obvious winner, bringing clear advantages in motivation and "classroom mood", code quality and, depending on pair composition, grades. Furthermore, since most students are forced to program in pairs anyway, it is very easy to implement.

Effective pair programming holds many benefits for students in computer science courses. Because the driver has the goal of building the program, drivers may not engage in dialogue as easily as navigators; however, the results presented by Rodríguez *et al.* (2017) suggests that more active participation from the driver may be important to improve learning. The findings show that feedback, information statements, and grounding may enhance the quality of the collaboration. Computer science educators can strive to enhance the effectiveness of classroom pair programming by encouraging active conversational participation from both partners.

Harsley *et al.* (2017) compared the interactions of student programmers working as individuals and as pairs to solve coding problems using an Intelligent Tutoring System. They analyzed over 79,000 log events from 173 students to explore on a large scale pair and individual programming interaction in fine-grained detail, including timing, example usage, compile success, and coding efficiency. They used unpaired t-tests to discover statistically significant differences in interactions between individuals and pairs. They discovered that pair programmers used fewer system-provided examples, completed problems faster and more efficiently, and showed higher indications of engagement with the task. Also, the majority of students in each condition reported satisfaction with the activity. Just as importantly, in both conditions, students experienced significant ($p < .01$) learning gains.

According to Issaee *et al.* (2021), pair programming contributes to making learning to program more effective, engaging, and inclusive. Issaee *et al.* (2022) also found that pair programming is an effective learning method for programming that additionally helps students acquire various other 21st-century skills in computer science classes.

2.2.7 Distributed pair programming or remote pair programming

Pair programming works remotely on the view of Bandukda and Nasir (2010) they observed that distributed pair programming, where pairs are in different physical dependencies communicating each other by voice and text messaging software, could be used to increase development productivity and the quality of the software product. A paper by Williams and Stout (2008) described some modifications in the agile practices adopted by a company, with teams in North America and Europe, to reduce problems in the development of software. They concluded that it is possible to distribute and scale agile teams, but keeping them small and together still is the best alternative.

In the systematic literature review of da Silva Estacio and Prikladnicki (2015) about distributed pair programming concluded that there is a need for further studies in the software industry on distributed pair programming. There is a need to investigate the effects of distributed pair programming on professionals. The level of dispersion of the teams is noted very well contextualized by researchers. There is an opportunity to research the effects of coordination, communication, and

cultural diversity on distributed pair programming. There is a need to explore more guidelines for the implementation of distributed pair programming in the software industry. There is an opportunity to explore the design, implementation, and tools for distributed pair programming. There is an opportunity to investigate distributed pair programming with other forms of collaborative programming.

D'Angelo and Begel (2017) describe a study of remote pair programming. Through observations, they identified that remote pairs would have significant difficulties communicating about on-screen code locations. They took advantage of cheap eye-tracking technology to develop a novel, dual eye tracking, gaze awareness visualization that helps pairs to see what the other person is looking at in a code document. They evaluated the visualization in an experiment in which pair programmers refactored source code while working in a simulated remote work environment. When the programmers used the visualization, they were more likely to look at the same thing at the same time. They were both faster and more successful at communicating using implicit, deictic references, and used them more frequently compared with references to the explicit line number or specific words on the screen. Most participants recognized the value of the visualization, though some were not even consciously aware that the visualization had changed their behavior and helped them communicate efficiently.

Rajpal (2018) examines a project where distributed pair programming was extremely effective. Contrary to agile dogma, some individuals may perform better in a distributed manner over collocated.

According to Adeliyi *et al.* (2021), students felt significantly more positive using Remote Pair Programming instead of face-to-face pair programming. As universities strive to provide a good student experience during a pandemic, these findings highlight the value of active remote pair programming using standard online communication tools.

We noted that distributed pair programming is the pair programming subtopic with more open questions nowadays. Open source projects and home-office work are remoting work, thus, is a relevant problem to be investigated.

2.2.8 Hybrid pair programming

Past research observed that some student pairs could contribute drastically unequal amounts toward their assignments. Yuan and Cao (2019) addresses this through a pilot of a hybrid pair programming model. Where students complete two halves of the assignment as independent programmers and then in partnerships. The results were that the hybrid model is easily adopted in Computer science classes and positions students to better face the obstacles presented by the assignment while allowing them to benefit from pair programming.

2.2.9 Distributed Pair Programming

Considering the ever-evolving trends towards global collaboration, distance education, and teleworking in the post-peak era of the COVID-19 pandemic, according to Xu and Correia (2023), programmers commonly work in a geographically distributed manner, and the demand for Distributed Pair Programming will continue to grow, especially as computational thinking becomes a fundamental 21st-century skill. Computational thinking enables real-world problem-solving and more accurate problem formulation leading to better solutions.

2.2.10 Camaraderie and personality

According to Li *et al.* (2013) who did an experimental study on pair programming to individual programming. The result observes pair programming increase the camaraderie. It also worked how a protective factor for preserving and reducing the evasion of women and novice programmers in computer science courses.

According to Hannay *et al.* (2010), personality could be a valid indicator of team performance in the long term. In a short time, it pays off for industry and researchers to focus on other performance preceptors, including experience and task complexity. Attention to the factors that may be relevant, such as team learning, team motivation, and programming skills. The reason is that personality traits are fixed in a person and could be focused on malleable factors, such as learning, satisfaction, and commitment to improvements to software development.

The influence of personality traits of programmers in pair performance is an important issue that cannot be ignored pair performance when working from same and remote locations according to Poonam and Yasser (2018). It was found that there is a relationship between personality traits of programmers, the location of programmers and pair performance. In general, the effect of personality traits is only significant when collaboration takes place from remote locations. Due to several limitations uncovered during the study, further investigation is required to obtain a holistic understanding for the effect of human factors on pair performance.

The results from Noori and Kazemifard (2015) showed that personality plays a role in the formation and utility of a pair. E.g., when the expertise of both individuals in the duo is high, the best pairing is introvert-extrovert. When both individuals are extroverts, the best pairing is low-high or medium-high expertise.

2.2.11 Dissemination of knowledge, quality, and reduction of defects

Williams and Kessler (2003) *apud* Brooks (1975), 'Adding manpower to a late software the project makes it later'. Adding a new member to the team that does not know the system creates the problem of the first contact with the code.

According to Williams and Kessler (2000) and Williams and Kessler (2003), the pair programming contributes to solving the problem of the first contact with the source code because the new team member can pair with someone more experienced.

To help spread the knowledge among the team, using pair rotation is a good idea. Thus, is reached a balance between pairs. As a side effect is the sharing of the expertise and competencies among the team members, according to Williams and Kessler (2000) and Cockburn and Williams (2001) and Nosek (1998).

Swamidurai and Umphress (2014) found an increase in productivity when the pair programming is dynamic when there is a rotation of the pairs in the team. Productivity is lower when paired programming is static, without team rotation.

The roles and pairs could continuously be exchanged, aiming to enable all team members to be knowledgeable about all parts of the system. Thus, it solves the problem of a single developer has an exclusive knowledge about a part of the system. By disseminating knowledge, more people participate in the development of a feature and learn how the system or a development tool works Beck and Andres (2004).

Constantine (1995) has published about the reduction of the number of defects when programming in pairs. Nosek (1998) found that pair programming improves code and algorithm quality, but also noted the increased development effort, as it observed that the pair spent more working hours than if a single programmer worked alone.

di Bella *et al.* (2013) published a case study to evaluate the effect of pair programming on the quality and efficiency of defect corrections. The software development project studied in the paper was a large Italian company for a period of the fourteen months. Other case studies done about pair programming took a shorter period comparing to this one, what makes this more realistic. The pair programming effectiveness in the context of defect corrections and implementations of user stories. The analysis points out the introduction of new defects using pair programming tends to decrease. The results are consistent for both contexts.

2.2.12 Small project, many developers and short period of development

The research conducted by Smith and Menzies (2002) analyses the problem of Pair Programming productivity and observes that NASA (National Aeronautics and Space Administration) should not use the agile method based on Pair Programming in 64 % of its projects. However, pair programming could be an advantage in specific cases: when the project is relatively small, when there is an abundance of developers, and if the development period is short.

For Padberg and Muller (2003) Padberg and Müller (2003), the code produced using the Pair Programming is more defect-free. They conclude, when market pressure is strong, to add developers and to create more pairs, can accelerate the project and can increase its value for the business, but it also increases the cost.

2.2.13 Informal review, inspection process and communication

Pair programming acts as an informal review process because each line of code written will read by at least two people. Code inspections help a lot in finding a high percentage of software errors. However, they are time-consuming to organize and often represent delays in the development process according to Williams *et al.* (2000).

Pair programming is a less formal review process and probably does not encounter as many errors as code inspections, however, is cheaper than a formal inspection process according to Cockburn and Williams (2000) and Cockburn and Williams (2001).

Cockburn and Williams (2000), Cockburn and Williams (2001) and Williams *et al.* (2000) observed that productivity when using pair programming, seems to be comparable to that of two people working independently. The reason is when pair programming, the developers will discuss the system before developing it, thus, probably will occur a fewer number of errors and less rework. The errors avoided is such that less time is spent repairing discovered bugs, during the testing process.

Du *et al.* (2015) performed a basic programming experiment using the C programming language. The result was analyzed and carried out interviews after this experiment. The conclusion is pair programming was effective in improving the communication of the students.

2.2.14 Keyboard ownership

It is important for the pair to take ownership of the keyboard to keep both programmers engaged Vanhanen and Korpi (2007) Williams and Kessler (2003). When the pair has a different level of experience, the more experienced programmer tends to dominate the keyboard ownership. Software engineers seem more engaged/committed doing the role of pilot/driver, with the possession of the keyboard. To promote the commitment of both, it is suggested to use two keyboards and two mouses Chong and Hurlbutt (2007) Freudenberg *et al.* (2007) Höfer (2008).

Plonka *et al.* (2012) study pair programming with professional programmers. They observed among the pairs, who were not in possession of the keyboard often had a reason to go away, that happened depending on the activity of those who are in possession of the keyboard, due interruptions or because they shared the work and done in parallel. They also observed the separation was not necessarily a problem.

2.2.15 Preceptors of performance and organizational culture

According to Hannay *et al.* (2010), personality in the future could be a valid indicator of the team performance. However, was not found a connection with pair programming in a consistent way related to personality, especially when includes indicators related to skill, task complexity, and country. In the short term, it pays off for industry and researchers, to focus on other performance preceptors, including experience and task complexity. Attention to the factors that may be relevant, such as team learning, team motivation, and programming skills. In contrast to personality traits

fixed in a person, there are others malleable factors as learning, motivation, and commitment to perform improvements to the software produced.

Plonka and van der Linden (2012) identified in four companies aspects that make difficult the adoption of pair programming. These aspects are related to organizational issues such as organizational culture and management style. Thus, they conclude about organizational culture could be considered a relevant factor for the expansion of pair programming adoption in the teams of these four companies studied.

2.2.16 Interruptions and cognitive learning

Sillitti *et al.* (2012) observed that people significantly change their work habits when pair programming and tend to focus more on productive activities. The work sessions are longer and uninterrupted, programmers focus more on productive tasks, reducing distractions such as emails, internet browsers, and chat programs. At the same time, the discussion is encouraged improving the solution created, and the partners helping each other to have a clearer understanding of the problem to be solved.

Jones and Fleming (2013) carried out a qualitative study of 7 pairs (14 students) during code-debugging tasks, emphasizing issues such as partner teaching, the contributions of the partner without the possession of the keyboard, and impact of partner interruptions.

Jones and Fleming (2013) observed in the results that all the pairs presented episodes of the teaching of the partner. The co-pilot contributed to introducing several productive ideas without any indication toward these partner interruptions were detrimental to the working flow.

Plonka *et al.* (2015) studied pair programming as a strategy for knowledge transfer between senior and novice programmers. This strategy is one of several teaching methods of Cognitive Learning.

The senior developer verbalizes his lessons when is piloting and when the novice is piloting is supported by the senior, reinforcing the cognitive learning.

Reflection (compare their solution with that of others) and Exploration (tasks for the novice to advance learning) also are teaching methods used in cognitive learning. The organizations usually ignored the teaching methods because their prioritization of the needs to produce code instead of the apprenticeship of programmers.

There is a gap in the verbalization of the novice during the learning process was not stimulated, even though it would improve the learning process. While seniors make an additional effort to transfer knowledge to the newbie by raising awareness about knowledge transfer practices, it generates learning opportunity for the senior, verbalizing is a teaching method used in cognitive learning according to Plonka *et al.* (2015).

2.2.17 Relation with user experience design

Seyam and McCrickard (2015) using students of computer science have researched about pair programming working together with human-computer interface requirements. They observed to beginner students using pair programming is more productive when the pair selection is not random, it is better for students to choose pairs or the pair be formed based on their grades in the programming class. The students should be encouraged to think about the project and how the work will be done, before start programming. The introduction of quality as a fundamental requirement rather than a bonus is better. Emphasize the practice of speaking and listening is better to evaluate through a programming challenge.

They contend that pair programming contributes to greater interaction among developers, which allows for better human-computer interface design and better user experience when using the software they developed according to Seyam and McCrickard (2015).

The Seyam and McCrickard (2016) in-class study showed that activities in a mobile development course would differ than regular CS programming course activities in several aspects. Issues related to mobile UI/UX, sensors, and multi-device connectivity introduce unique challenges to students

in the mobile development classes. Pair Programming seems to provide a reasonable approach to handle such challenges, though with changes put in place. For Pair Programming to perform better in mobile development classes, instructors should consider how Pair Programming has traditionally been applied, as well as why some changes may positively influence student learning and performance.

2.2.18 Helpful for big data/data science projects

The case study of Saltz and Shamshurin (2017) explored the use of pair programming in a data science context. While much has been explored with respect to pair programming for software development, very little has been reported with respect to pair programming for teams doing data science activities. Their case study focused on R pair programming data science activities. Perhaps not surprisingly, they found pair programming to have similar benefits as has been found within a software development context.

In particular, their case study revealed that pair programming is effective for data scientists to improve communication and create better code in less time. Hence, Saltz and Shamshurin (2017) observed that pair programming is helpful for big data/data science projects.

2.2.19 Productive expert-expert pair

Zieris and Prechelt (2016) have shown that pair programming is not either productive or a knowledge transfer technique, but both at once. They qualitatively analyzed several in-vivo pair programming session recordings of professional software developers. They illustrated their findings using one full-length session of a productive expert-expert pair with many knowledge transfer episodes. The main takeaways are:

- Pair programming is useful in more cases than one might have thought. Even if you have a highly skilled developer who could tackle tasks on her own, collaborating closely with another expert can be beneficial.
- Knowledge transfer is an important element even of pair programming sessions with two expert participants.
- Do not focus on wall-clock time alone. Even though they are hard to pinpoint, arguably, the effects of successful knowledge transfer reach far beyond a single session.
- Even in expert-expert constellations, not every knowledge transfer episode goes frictionless. There is a clear potential for training software developers to become better pair programmers.

2.2.20 Different brain activity was observed between pair programming and solo programming

Busechian *et al.* (2018) are claiming since software development involves a significant amount of mental activities, a full Electroencephalography can be successfully applied to evaluate the effectiveness of various software development processes and practices. Moreover, they claim this is now becoming effectively doable, due to the availability of portable Electroencephalography devices. To this end, they have carried out an early, practical experimentation to perform a preliminary study on the impact of pair programming on brain activity. The result of Busechian *et al.* (2018) early experimentation is that the approach is, indeed, doable and that the outcome of the experimentation identifies a very different brain activity between pair programming and solo programming.

Thapaliya (2020) measures the concentration level, whether it is high or low, under three different cases: solo programming, pair programming (navigator), and pair programming (driver). The preliminary results confirm the higher concentration level as compared to solo programming.

2.2.21 GitHub Copilot vs Human Copilot

Through an experiment with 21 participants, Imai (2022) focuses on code productivity and code quality. For experimental design, a participant was given a project to code under three conditions presented in a randomized order. The conditions are pair-programming with Copilot, human pair-programming as a driver, and as a navigator. The codes generated from the three trials were analyzed to determine how many lines of code on average were added in each condition and how many lines of code on average were removed in the subsequent stage. The former measures the productivity of each condition, while the latter measures the quality of the produced code. The results suggest that although Copilot increases productivity as measured by lines of code added, the quality of code produced is inferior by having more lines of code deleted in the subsequent trial.

2.2.22 Summarizing the categories founded in the main topics researched and the main strengths and weakness of Pair Programming

The Figure 2.2 is a cognitive map resuming the categories presented in subsections 2.2.1 until 2.2.21 to help recall important features.

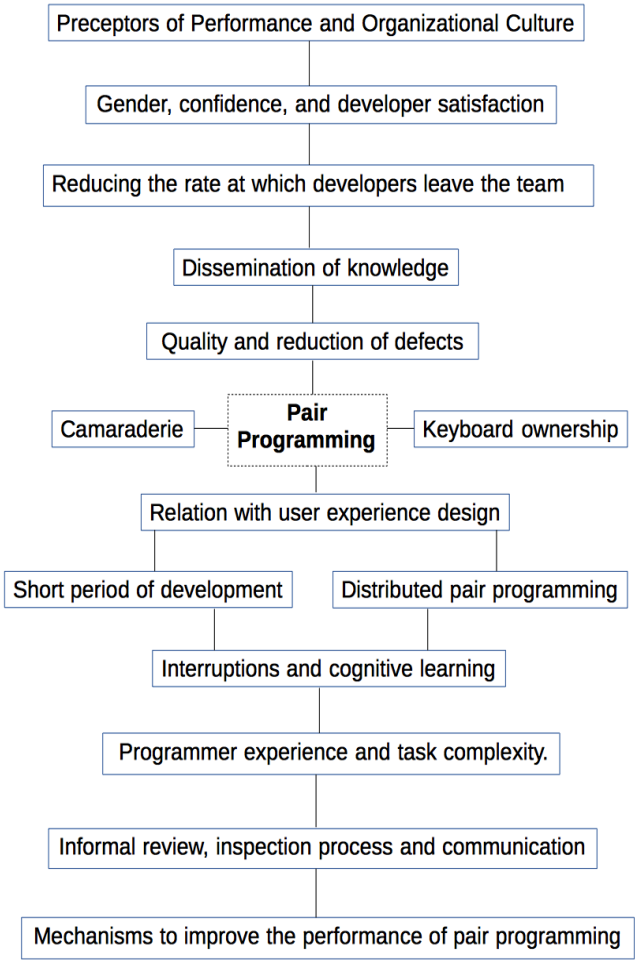


Figure 2.2: *A cognitive map summarizing the categories founded in the main topics researched about Pair Programming presented in subsections 2.2.1 until 2.2.21*

Table 2.2 summarizes the main strengths and weakness of Pair Programming presented in subsections 2.2.1 until 2.2.21 in a chronological way.

Year	Strengths	Weaknesses
1975	Reduces defects	
1995	Pair development Pattern	
1995	Programmers' confident	
1998	Improves algorithm	
1998		Increase effort
1999	Improves quality	
2000	Communication	
2000	Collaboration	
2000	Team members' trust	
2000	Team members' satisfaction	
2000	Sharing of knowledge	
2001	Refactoring stimulation	
2002	Good for small teams	Bad for large teams
2002	Programmers' abundance	Programmers' lack
2003	First code contact	
2003	Pair rotation	Navigator' focus
2003	Time-to-market reduction	Increase cost
2003	Increase business' value	Increase cost
2004	Programmers' motivation	
2004	Member' exit risk reduction	Loss of productivity
2005	Gain of productivity	Require proper design
2005	Complex tasks	Simple tasks
2006	Students' evasion reduction	
2007	Design and complex tasks	Awkward to experts
2007	Two keyboards	Junior only watch
2009	Large and complex projects	
2010	Personality's small importance	
2010	Pairs works remote	
2010	4 ways to improve	Trim productivity loss
2012	Focus on Productivity tasks	
2012		Require culture change
2012	Separate work sometimes	
2013	Increasing learning	Chat considers gender
2014	Cognitive learning	
2014	Rotation on pair selection	
2015	Simultaneous Style Pair Programming or PRSP	
2016	Expert-expert pair can be productive	
2017	Helpful for big data/data science projects	
2018	Different brain activity between pair and solo programming	
2019	Hybrid Pair Programming	
2020	Higher concentration level as compared to solo programming	
2021	Remote Pair Programming	
2022	GitHub Copilot vs Human Copilot	
2023	Distributed Pair Programming	

Table 2.2: *The main strengths and weakness of Pair Programming presented in subsections 2.2.1 until 2.2.21 in a chronological way*

2.2.23 Simultaneous Style Pair Programming

A possibility to increase the productivity of pair programming is the use of parallelism and multidisciplinary teams of Concurrent Engineering of Pithon (2004). Another alternative is the incorporation of a process of pair code review, and a stage for projecting pairing simultaneous development. In the pairing project, whether or not adopting pair/mob programming is defined collaboratively and if it is adopted, define in which software requirements each programming technique will be used.

As Coplien and Harrison (2004) argue, the design is adopted compatible with the pairing of working together. In this way, they can produce more than the sum of the two individually. The selection of pairs depends on the project, the task to be performed, the availability of the team members, the need to disseminate knowledge, the pair rotation and the experience of each one.

When incorporating Concurrent Engineering practices into pair programming, one should note the importance of communication, even if the activity is performed in different physical dependencies. It is also important to stress that the more the activity is developed simultaneously, the greater the productivity gain.

Communication and collaboration are the pillars to join the work performed simultaneously on different computers successfully, without compromising with mistakes during the joining of the work, the productivity gain obtained by the parallelism during its execution according to Pithon (2004).

Developers work in pairs to accomplish their tasks to promotes collective, collaborative work, uniting the team, improving communication and the quality of the code. The work is developed simultaneously and if there is more than one team pair, the iterations are designed to be simultaneous.

The Programming and Review Simultaneous in Pairs (PRSP) of Kattan (2015) is also known as Simultaneous Style Pair Programming or as pair development. PRSP has the following definition:

> "A programming activity wherein planning is at the beginning including the pair selection, the pairing of tasks is collaboratively designed and based on this two programmers work collaboratively in the same activity.
>
> Only in the beginning of one activity sitting side by side to exchange experiences (this way there are more algorithms and solutions) or communicate in the beginning, if they are working in a distributed way (different locations).
>
> Still, in this initial phase, they decide how to divide the task, and do not need to sit together all the time on a single computer, or communicate at all times if they are working in a distributed way, only when necessary and useful.
>
> Whenever possible, the work should be performed simultaneously on separate computers. Unlike traditional pair programming, in PrsP each programmer revises the work of the other one simultaneously using two computers if an error is found then the task returns to the programmer fix it.
>
> In the end, they unite the work of the pair. During the rest, it is suggested to speak or to think about the best way for the accomplishment of the work, mindset zero defect, adoption of a process of stress reduction and for resolution of conflicts".

Groupware can be used to automate, control, track and verify through software the procedures performed by the multidisciplinary team in the development of collaborative and simultaneous work.

It may be advantageous if the team values or needs to control or track the activities of the pairs, as well as to obtain data for some metric used in some indicator chosen by the team, but agile teams typically give up control for valorizes communication and trust. Zero-Defects Mindset used for quality.

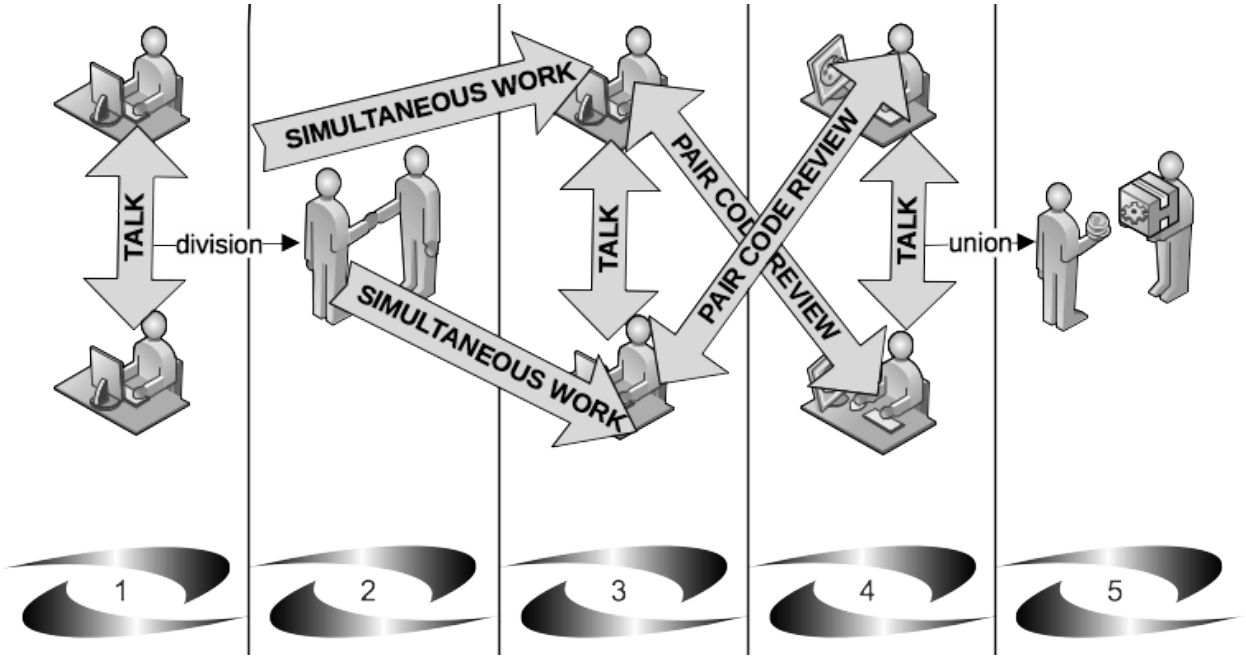


Figure 2.3: *Drawing about the five phases inside of the Programming and Review Simultaneous in Pairs. The total is seven phases, the first for planning and the last for productive rest are not at drawing.*

The PrsP is one extension to the pair programming. It is proper when the goal is to reduce the time-to-benefit. Figure 2.3 illustrates the phases of PrsP, excepting the planning phase and reflective/productive rest phase because they have a miscellaneous of possibilities make them very hard to draw.

2.3 Mob Programming

The idea of mob programming originated from technical meetings where a team member presented a code he knew. The group could work together, exchange information on design, architecture and programming techniques, encourage change in the code and provide feedback according to Hohman and Slocum (2002).

The main difference, comparing to pair programming, is that the whole team works together as part of the pairing. In addition to software coding, Mob Programming teams works together on almost all tasks that a typical software development team tackles, such as defining stories, designing, testing, deploying software, and collaborating with the customer Zuill (2014).

According to Dalton (2019), Mob programming is a technique used by a collaborative software development team to fast solve a problem or to write complex software. Mob programming is similar to pair programming but with two distinctions. First, mob programming uses as many developers as possible so that many perspectives lead to a higher complete solution. Secondly, mob programming is performed on demand. Moreover, it is not a standard behavior for the team.

Mob Programming is a natural extension of the popular Pair Programming concept according to Pearl (2018), and is not restricted to a specific programming language or technology. It can be used by anyone who develops software, including dev leads, software developers, and agile coaches. The more people working on a bug or feature results in fewer dependencies on individuals, and overall increased learning for everyone involved. With more eyes on the code, you'll find you develop better solutions with fewer defects.

The whole team works around a single workstation, only one member has the possession of the keyboard to modify the source code Zuill (2014). Mob Programming can be observed as an evolution of the learning environment in a Coding Dojo setting to a production environment Bravo and Goldman (2010).

Mob Programming, according to Dragos (2021), is the preferred practice for solving issues that have flexible acceptance criteria and requirements. By using Mob Programming, the collective domain knowledge is increased and can drastically reduce the time required to adapt to a change. Another case in which Mob should be chosen over Pair is when implementing new features or services. In this case, having multiple perspectives and arguments when making important design or implementation decisions can increase the quality and longevity of the feature/service.

If everyone is new to the IDE, one person may be navigating, another driving, and someone may be researching. The researcher may find the auto-format key and tell the team. In this example it is easy to see how improvements naturally make it into the system for the entire team, Lucian (2021) explained. Since everyone has different experiences and preferences, you get the best of the combination of habits, and people begin to share habits with each other, he said at Agile 2021.

Among the difficulties in adopting the mob programming, Hohman and Slocum (2002) report that it is difficult to keep in focus, especially for those who are not with keyboard possession. According to Wilson (2015), there is the concern of managers about the difficulty of having the whole team working together in a single requirement, in the view of them seems to be more productive to have pairs working on different tasks. On the other hand, Zuill and Meadows (2016) highlights the benefits of communication, collaboration, and team alignment.

The troublesome part about researching about mob programming according to Shiraishi *et al.* (2019) is to obtain data and results from mob programming sessions. Since a mob programming session take few months to complete, it is time consuming. That as well as to discover multiple individuals (more than 3) to convince them cooperate for more than 1 month is also an obstacle.

Another important aspect is the configuration of the room. Every team member with own chair to move around on the different roles of driver or navigator. Thus, the people do not need to constantly adjust chair settings so each one can stay as comfortable as possible.

The emergence of Mob Programming, where diverse teams cooperate closely to build a software product and take a vision towards Agile adaptations according to Müter (2018). Beyond that, by Rashi Agarwal (2020), a system needs to be adaptive by understanding the changes occurring in the needs of the application user.

Tens and hundreds of interactions between people occur every day in their work. The people express ideas, discuss problems, explore possible solutions, and share thoughts all day long. To make it possible and to keep this high level of communication happening throughout the day, the team must adopt the principle of treating each other with kindness, consideration, and respect according to Zuill (2014).

The room has a rolling magnetic whiteboard for keeping track of the work. Retrospectives are used in Mob Programming sessions. In these ceremonies, a team frequently evaluates what is working for them, what problems they might be having, and how they can improve. These are usually short and focused on one item. Figure 2.4 of Zuill (2014) and Zuill and Meadows (2016) illustrates this basic floor plan, is a setup that worked very well and had been seen it at some companies according to Zuill and Meadows (2016). A possibility is to have one, two, or more monitors. The readjustment of the monitor height to be a height suitable for everyone.

Retrospectives are used in Mob Programming sessions according to Zuill (2014). In these ceremonies, the team frequently evaluates what is working for them, what problems they might be having, and how they can improve. These are usually quick and focused on one item.

There are two roles: the Driver, and the Navigator. The Driver has the possession of the keyboard. The Navigators discuss the idea being coded and guide the Driver in creating the code according to Zuill (2014).

A possibility to Mob Programming from Pyhäjärvi and Falco (2016) is adopting the Driver/-Navigators pattern adapted from Llewellyn Falco's "strong" pair programming style. The basic rule is for an idea to go from your head into the computer it MUST go through someone else's hands.

The team adopting Mob Programming has the possibility of a timed rotation. Each team member plays the role of Driver with the possession of the keyboard for a short time. The timer remembers the current driver when can pass the possession of the keyboard to the next driver when their turn

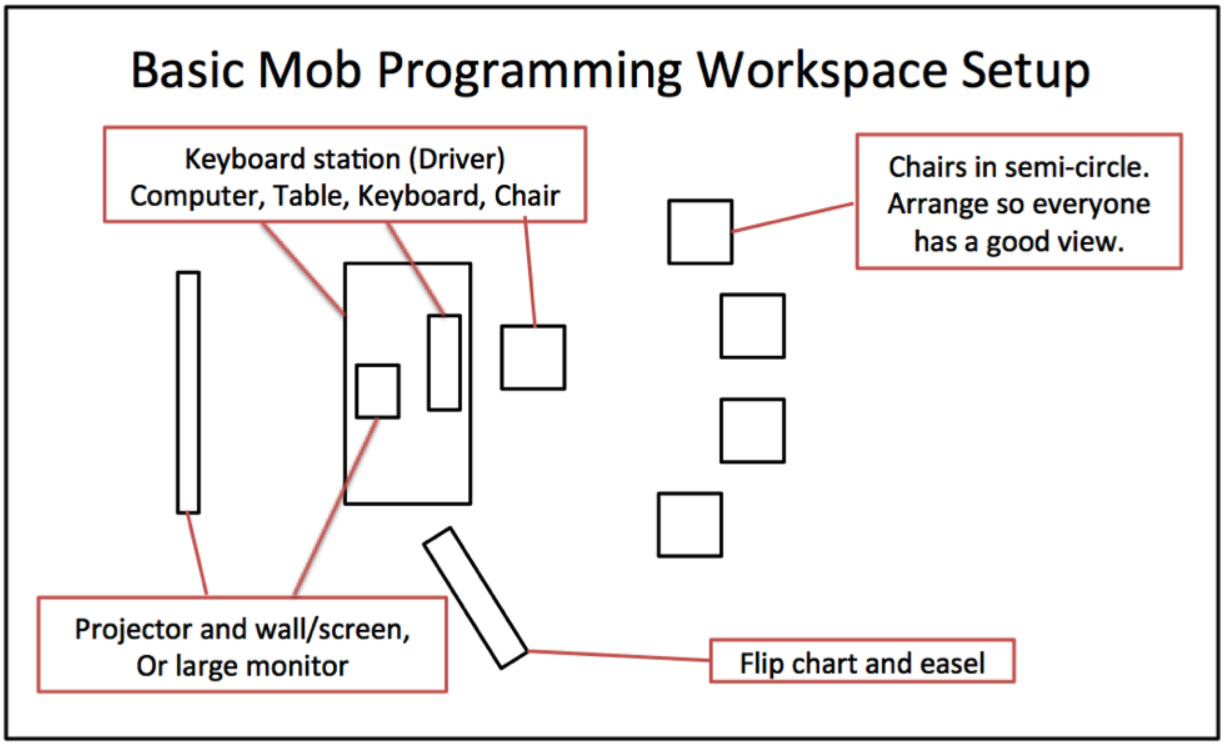


Figure 2.4: *This setup has been found to work well and has been seen at several companies. You can have one, two, or more monitors. Experiment with monitor height to make sure it is at a height that works well for everyone, illustration from Zuill and Meadows (2016).*

ends up according to Zuill and Meadows (2016).

Concerning weaknesses, Hohman and Slocum (2002) report the difficulty to keep focus for the co-pilots/navigators without the possession of the keyboard. Wilson (2015) cites two problems, the effect of dominant personalities within the Group and the appearance of Groupthink.

Among the strengths, Zuill (2014) and Zuill and Meadows (2016) emphasizes that it strengthens communication, team alignment, collaboration and self-organized. Wilson (2015) observe an increase in the dissemination of knowledge and the technique worked better for critical and complex software products.

There are two points of convergence in the conclusion of Hohman and Slocum (2002) and Wilson (2015) when using two projectors/monitors Mob Programming works better and is advantageous to switch the use with other techniques. If the main problem derived from the team not having a Shared Understanding of the project, Mob Programming involves an entire team is a great approach according to Zuill (2014), Wilson (2015), Boekhout (2016), Hohman and Slocum (2002), and Lilienthal (2017).

The initial evidence for the effectiveness of mob programming by its early adopters is quite encouraging, it still requires validation through rigorously designed empirical research, so that more organizations could adopt it fully understanding the conditions that accentuate its benefits and minimize potential risks Balijepally *et al.* (2017) and Kattan and Goldman (2017).

The room must be physically comfortable while the team members work relatively close to each other, using shared monitors, keyboards, computer setup, and programming tools. There is only one computer in use for programming Zuill (2014). Figure 2.5 illustrates a setup that Laptop computer of the Driver permit to him see the screen, while the others team members watch on a large monitor.

Fun is fundamental in life since it fosters interaction and learning Vieira and da Silva (2017). In three case studies of open source software, all developers reported that had fun with Mob Programming and the main conclusion was an improvement of the learning experience Kattan *et al.* (2018a).

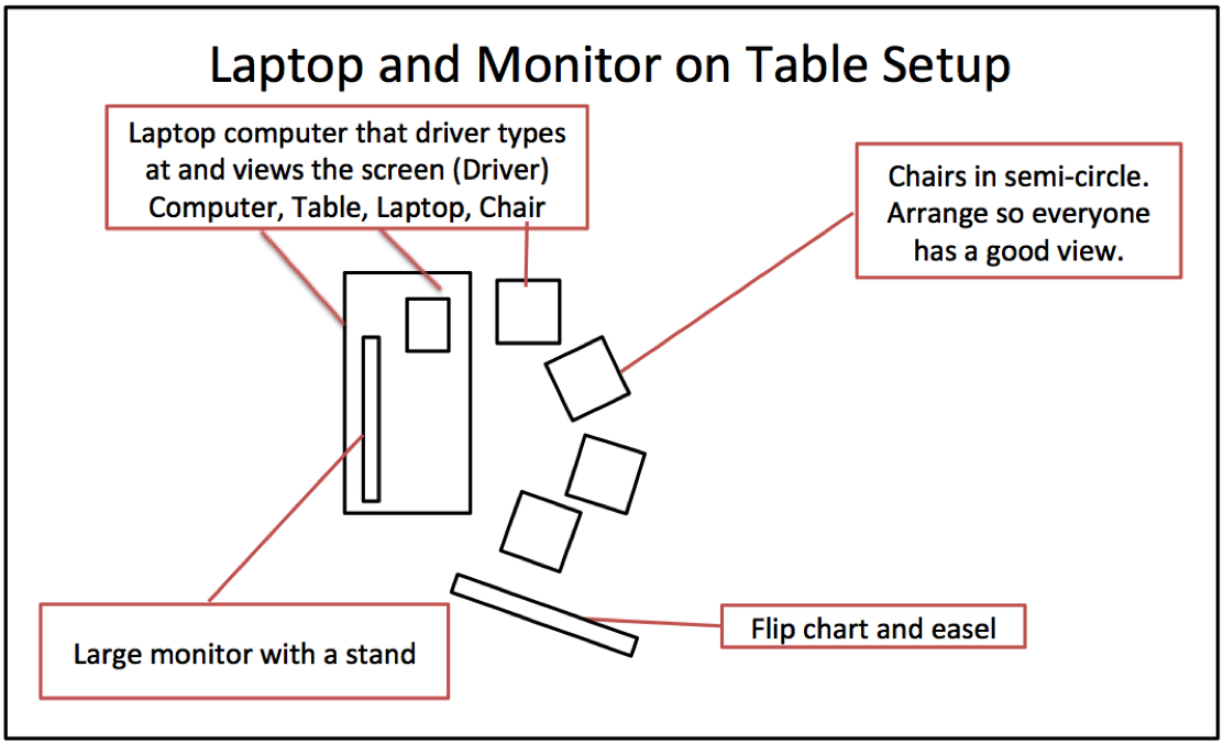


Figure 2.5: *This is a Zuill and Meadows (2016) illustration of a room setup that permits the Driver to see the screen simultaneously while the other Mob Programmers watch on a large monitor.*

Beyond the advantage of alternate with other techniques, we found some aspects of the use of Mob Programming over other techniques to sharing knowledge, learning, and satisfaction of the programmer. Additionally, collaboration among teams is improved and the bond formed among the team members Kattan *et al.* (2018a).

Mob programming is surprisingly inclusive. Done right, it can provide a safe and nurturing environment for the fledgling programming skills of novice programmers or new team members. Not only this, remote mobbing offers significantly more support and flex for family life or other interests. Most unexpectedly of all, the ability to uniquely tailor a physical environment that is right for you to work in would be almost impossible to achieve in a traditional work environment. Couple this with a level of optionality about level and duration of social that is also difficult to achieve in an office and could be a blueprint for maximizing inclusivity for the neurodivergent team members according to Freudenberg and Wynne (2018).

On the Buchan and Pearl (2018) team's experience and the survey results has suggested several areas that need further investigation. In what contexts is it better to code in a mob or a pair or solo? What are the perceptions of the mob team members on the value of mobbing? Are there any theories from other disciplines underpin mobbing practice and explain and predict outcomes? Some quantitative empirical evidence of the benefits of mobbing (or otherwise) would be informative.

According to Zuill and Meadows (2016) a set up of the room possible is two projectors or monitors. The goal is to keep the screens at about the same size, general position, resolution, and brightness to make them comfortable to work with all day long. There are also two keyboards and two mouses (a simple one and another ergonomic one).

Figure 2.6 is a photo of the team working in the programming, note that they are using a laser pointer, making easier to point out an issue in the source code to be fixed. They had two projectors available and computers to parallel searches Kattan *et al.* (2018a).

According to Zuill and Meadows (2016) everyone has the choice to suit him/herself in the better way. Figure 2.7 of Zuill and Meadows (2016) illustrates a setup that could be more comfortable in the view of some teams. The table could be a smallish circular or square. The team could use the



Figure 2.6: *The team is working on the programming. Note the Navigator is strategically positioned, and he has a laser pointer, making it easier to point out an issue in the source code to be fixed, photo from Kattan et al. (2018a).*

imagination because in the fundamentals of a Mob Programming is valued the liberty of choice.

Based on the findings from Björklund *et al.* (2020), the interviewees pointed out that the motivation towards work is affected by using Mob Programming. The ability within the team to motivate each other was something that stood out from the interviews as a key argument as it was a reliable source of motivation. An interviewee described their relation to motivation with Mob Programming as:

> "*I feel that the motivation gets better because, if you need a 10 min break or something, you can do it and there are still other members who continue to work. So, there is always someone who is motivated and who motivates others. Therefore, it is easier to be motivated when you need to help others than when you sit at the computer by yourself. So, I think it helps a lot that way.*"

Another interviewee from Björklund *et al.* (2020) referred to the flexibility within the mob regarding the continuous work done by the team:

> "*It's quite common that I need to go away and do other things, and then it is very easy to get into the task again when you are using Mob Programming. You also get a feeling that you are contributing even if you can't be with the team all the time.*"

The results from nine semi-structured interviews from Anderfelt (2020) suggest that beyond

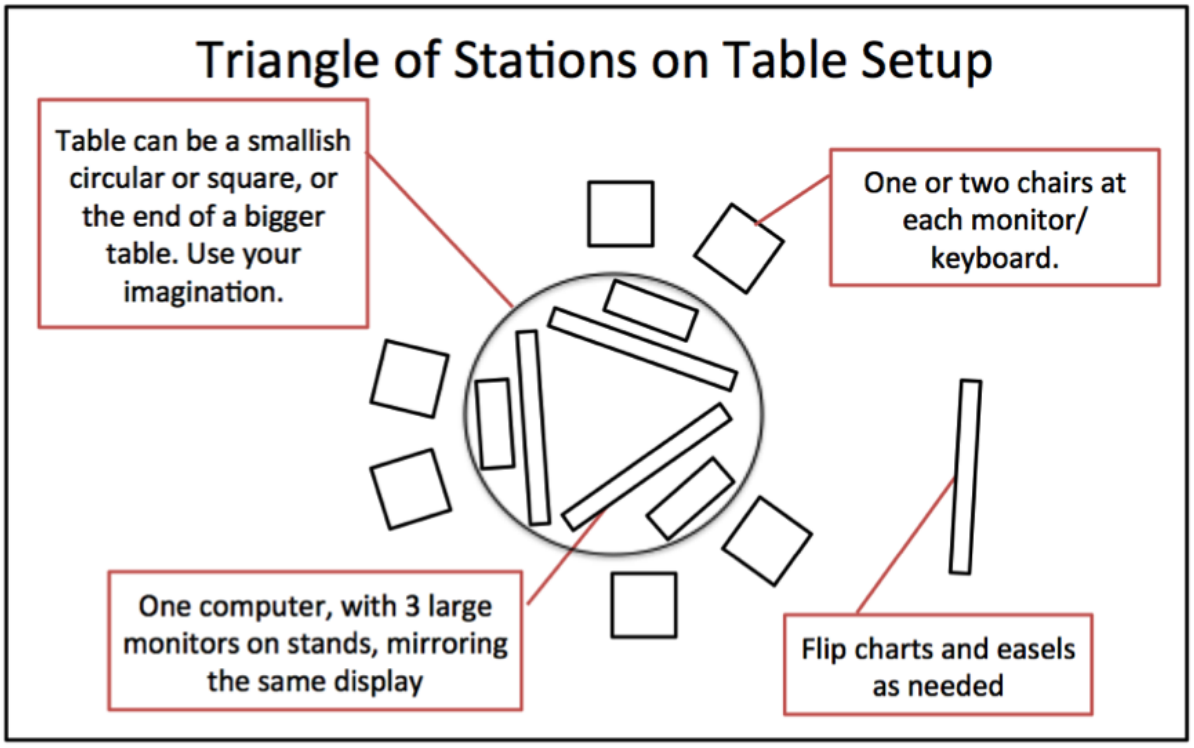


Figure 2.7: *The team could use the imagination because in the fundamentals of a Mob Programming is valued the liberty of choice according to Zuill and Meadows (2016).*

most previous studies show, mob development is a method that can be used for more than just programming.

There were 24 students participating in the study of Roque Hernandez *et al.* (2020) who were taking the subject of Intermediate Programming in the third semester of the undergraduate program in Information Technologies. The age range was between 19 and 25 years. The results showed that students prefer pair programming because they perceive it as a midpoint between programming alone and doing it with a large group. Solo programming may cause stress and intellectual blockage, and doing so with too many people at the same time may generate distraction and imbalance of work among the participants.

Mob Programming is a mostly unknown and rarely practiced team formation strategy that has the potential to boost communication and learning, among many other benefits, within a group of software developers, according to Stephenson (2023). Teams struggling with finding a successful way of working together are encouraged to try Mob Programming to see if its structured approach to team collaboration can help them overcome issues they experience. Discovering more about your team members can also aid in better rules of engagement regarding communication and empathy.

2.3.1 Different brain activity was observed between mob programming and solo programming

Umezawa *et al.* (2017) investigated the use of brain waves to estimate the degree of difficulty of a learning task. Task difficulty is measured by Shiraishi *et al.* (2021) using brain waves to estimate suggested by Umezawa *et al.* (2017). MindWave Mobile 2 was used to collect brainwave data from a new research perspective. The results indicate that group programming can realize more efficient learning than individual programming. The meditation levels were lowest in solo work which suggests working alone may lead to less stress and tension compared to group work.

2.3.2 Mob Programming and Pair Induce Higher Levels of Concentration (Attention) Compared to Solo Programming

In many situations Shiraishi *et al.* (2021) observed that tasks requiring high concentration are often worked on alone. In Figure 2.8, concentration (attention) levels were highest in the neutral state. Therefore, the average percentage change in each programming session (Figure 2.9) decreased concentration. Pair programming showed the highest percentage of attention level change. However, both group programming sessions (pair and mob) had higher concentration levels than the individual programming session.

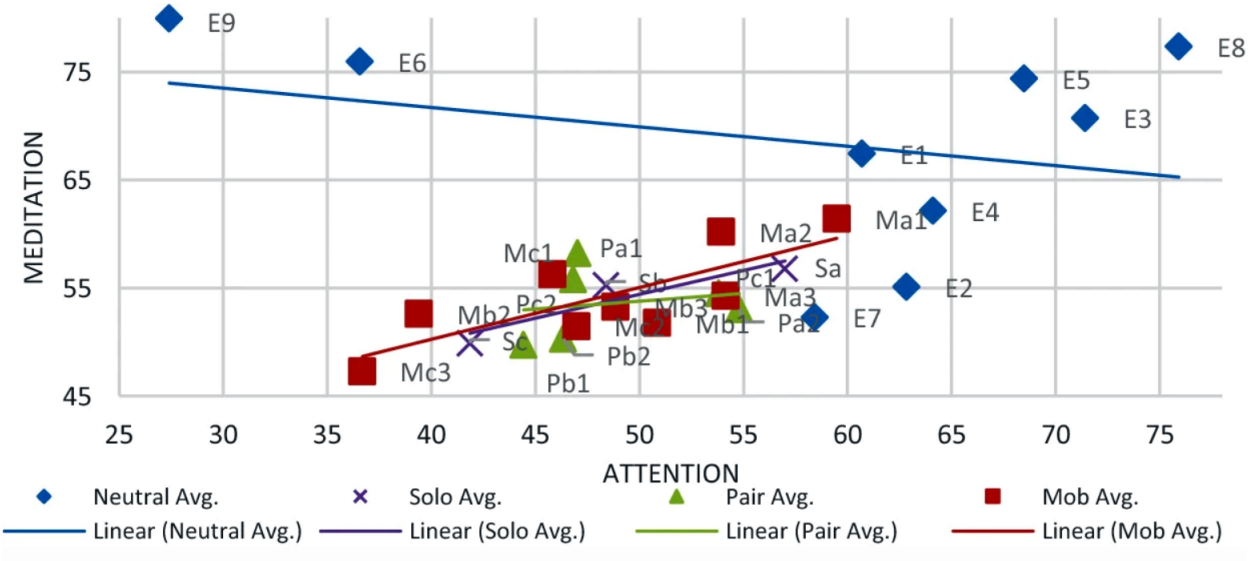


Figure 2.8: *Comparison of meditation and concentration (attention) levels by programming state from Shiraishi et al. (2021).*

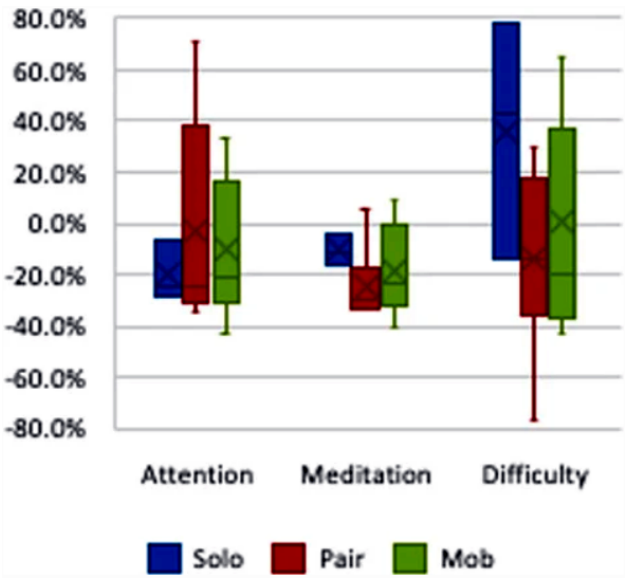


Figure 2.9: *Box and whisker of participants' average percentage in change of concentration (attention), meditation, and difficulty from the neutral state by programming session from Comparison of meditation and attention levels by programming state from Shiraishi et al. (2021).*

2.3.3 Group Sessions Induce Lower Levels of Calmness (Meditation) Compared to Solo Programming

In Figure 2.8 from Shiraishi *et al.* (2021), all participants except two showed the highest meditation level in the neutral state. Overall, a decrease was observed in the average for all three types of programming sessions. The highest average meditation level was solo, and the lowest was pair programming. Their difference was significant (around 13%). This result differs from a previous study from Busechian *et al.* (2018), where a more relaxed state was observed in pair programming than solo programming.

2.3.4 Difficulty Decrease When Working in a Group

Previous research from Laughlin *et al.* (2006) found that groups of three to five individuals perform best when solving complex problems, suggesting that groups are better suited than those working alone to address challenging problems. Additionally, those in a group tend to retain more information, especially since the group can work together and share knowledge. Finally, working with others provides new insight and perspectives. If a significant difference can be seen within individual/group size, we will be able to assign tasks to groups or individuals according to the difficulty of the task. Thus, making it much easier for participants to program a task.

The questions in each session had the same level of difficulty in the experiment of Shiraishi *et al.* (2021). However, differences in difficulty are observed by session type. The lower the difficulty value, the easier a task feels. Figure 2.9 shows the average change in difficulty by group relative to the neutral state. The solo experiment showed a 36% increase in difficulty, whereas the mob programming group had a 1% increase in difficulty compared to the neutral state. On the other hand, pair programming showed a decreased difficulty of around -14%. These results imply that working in a group setting is less difficult than working alone.

2.3.5 Online Courses using Mob Programming with a mentor role

According to Kaieda *et al.* (2021) higher education curricula have started implementing online courses as a method for teaching programming, which poses challenges when conducting XP. Because social factors are vital to learning, we examine a project-based learning method using online Mob Programming that introduces a high level of communication. It also maintains the advantages of Pair Programming through the 'driver' and 'navigator' approaches. However, there are multiple complications, including discomfort, slow code generation, and interpersonal issues, when using conventional Mob Programming. Accordingly, Kaieda *et al.* (2021) introduces a mentoring approach to the Mob Programming setup and examines the differences compared to conventional Mob Programming sessions. The results suggest that the proposed mentor method has a higher learning effect in programming education than the use of Mob Programming without a mentor.

The concern with group activities from Sreecharan Sankaranarayanan (2019) is that group processes could come in the way of learning and productivity, but without structure, they can descend into chaos. Without guidance, students can choose sub-optimal strategies, such as working on tasks or roles they are already good at, rather than taking the opportunity to learn. Additionally, group work can be dominated by the most outspoken members according to Johnson and Johnson (1984). In Sreecharan Sankaranarayanan (2019) work, the role of the facilitator would be performed by an intelligent conversational agent based on the open-source Bazaar framework from Adamson *et al.* (2014). Providing a structure around the collaboration and helping students understand how this structure contributes to their learning and success on the problem can stem group coordination difficulties at the outset. Assigning students to roles and periodic rotation of the role assignments can prevent the adoption of suboptimal strategies Hilton and Sankaranarayanan (2019). Moreover, the role assignments can mitigate the problem of social loafing by introducing an element of social pressure, but at the same time, balancing that pressure with support from team members in other roles. Students are exposed to different perspectives in solving problems, building solutions, experimenting, debugging, and writing readable code. They are also forced to externalize their thinking

which provides the opportunity for knowledge gaps to be revealed and addressed. Finally, they also have the chance to observe knowledge and expertise in action as they learn. This study will serve as a first step towards realizing these benefits of Mob Programming.

According to Kaieda *et al.* (2020) Mob Programming using a student mentoring method allows cooperative behavior between experienced and beginner students.

2.3.6 Remote Mob Programming

Christ *et al.* (2020) enjoy working from home: a quiet environment, perfect working setup, no commuting, and more quality time with their families and friends. In previous projects, Christ *et al.* (2020) sometimes felt isolated from their colleagues. This radically changed with Remote Mob Programming. It is real teamwork. Remote Mob Programming combines two ways of working: Mob Programming and working as a distributed team. With Remote Mob Programming, Christ *et al.* (2020) collaborate closely in the same virtual space. Following are some recommendations from Christ *et al.* (2020):

- **Remote Everybody**. To collaborate well in a distributed team, it is essential that everybody works remotely by default. It does not work if part of the team works on-site;
- **Camera Always On**. In a multi-monitor setup, Christ *et al.* (2020) make sure that the camera is at their main screen so that they're looking at each other. Christ *et al.* (2020) mute when they go away from keyboard, but leave the camera on;
- **Regular On-Site Meetings**. Christ *et al.* (2020) meet on-site once a month;
- **Small Team**. According to Christ *et al.* (2020) with Remote Mob Programming, only one person can talk at the same time. With larger teams, the individual speaking time shrinks, making it harder to stay focused. It is easy to become mentally absent. Also, expect technical issues, such as connectivity problems or interfering noise, to happen more frequently in a larger team. Those issues will block the whole team;
- **Screen Sharing**. Christ *et al.* (2020) feel most comfortable working in their environment. Christ *et al.* (2020) reports were more productive;
- **10 Minute Intervals**. Christ *et al.* (2020) rotate every ten minutes;
- **Group Decisions**. In Remote Mob Programming, all decisions are group decisions. Christ *et al.* (2020) minimize technical debt. With their wealth of know-how, experience, and opinions to discuss, they were able to make well-founded decisions everyone agreed with. When they were coding, they all agreed on changes and code styles. As a consequence, they don't need code reviews or pull requests;
- **Learn from the Team**. Sharing knowledge is at the heart of Mob Programming. Christ *et al.* (2020) get better every day by learning from each other;
- **Save the Planet**. Daily commuting causes traffic jams, crowded trains, and significant greenhouse gas emissions. Even worse, many consultants fly to their customers' offices. Christ *et al.* (2020) don't travel, so zero greenhouse gas emissions;
- **Dine with your Family**. As software engineers, Christ *et al.* (2020) often struggle to balance challenging and rewarding work with time for family and leisure. Sometimes, it feels mutually exclusive. Christ *et al.* (2020) enjoy more quality time with our families.

Turner (2021) has learned after one year of using Remote Mob Programming that many of the skills needed for in-person mob programming translate to remote mob programming. Many of the things necessary for successful in-person mob programming are also required for Remote Mob Programming, such as having adequate equipment and a mob timer. Some requirements for

successful in-person mob programming translate mostly unaltered to remote mob programming, while other requirements need to be adapted, such as ways to contribute to the code. Turner (2021) has also learned that some requirements for remote mob programming are mostly unique, such as ensuring that mob members have the correct tools to communicate. Turner (2021) has learned that remote mob programming is effective. Given the right tools and conditions, remote mob programming is as effective as in-person mob programming. Turner (2021) has learned that remote mob programming is a productive way to craft high-quality code and deliver valuable features to our customers frequently. Remote mob programming pairs well with technical and other practices, such as test-driven development, continuous integration, push button deployments, limiting work in process, and eliminating waste.

2.3.7 Mob Programming during COVID-19

In corporate environments, implementation of Mob Programming has shown that a team working together in a conference room can produce products faster than individuals working in their cubes and offices. During Covid, this technique has been expanded to include teams working online through their Zoom / MS Teams / Google Meet environment. Efforts to transition Mob Programming into virtual collaborations opens this technique to ready application in the astronomy field, where code is often either used and adapted by strangers across multiple locations, or is written by myriad students and postdocs who each have dominion over their own specific functions and features. Gay *et al.* (2021) describe how Mob Programming has been adapted for open source software development by the CosmoQuest team and is being used to engage professional software developers as volunteers in astronomy software projects. This online collaboration approach produced more robust software and allowed modern coding techniques to flow freely from industry to astronomy research.

2.3.8 Lead time optimization and productivity when applied to complex tasks covering multiple domains

In summary, Stahl and Martensson (2021) find that different types of tasks clearly differ in suitability for mob programming: interview results indicate that mob programming has positive effects on lead time optimization and productivity when applied to complex tasks covering multiple domains, whereas effects on productivity more generally are ambiguous and likely to be modest. Similarly, implications of mob programming on learning are ambiguous: though frequently raised as a benefit in literature, it is possible that it may lead to a more passive learning style. Individuals react differently to mob programming, however, not least with regards to exhaustion and to the social dynamics within the mob. This implies the need for organizations to be flexible in their adoption of the practice, allowing developers to participate depending on preference and ability. Unless properly managed, the these individual differences may give rise to tension and the forming of teams within the team.

2.3.9 Group Size

Powell and Anslow (2019) now up to 11 mobs, averaging 3 to 5 people per group. Other departments within Clearlink, including their digital outreach team, have taken notice and incorporated the principles of mob programming techniques into their work.

Each Mob Programming group, on average, consisted of four people at Hunter Industries. To grow their capacity, initially, Lucian (2017) hired four members in each Mob. However, they found that five typically is better to facilitate people taking vacations and dealing with possible distractions. With the assumption that any feature will have four people working on it, the developers self-organize around available work.

2.3.10 Scaling up to multiple Mob Programming groups

As Lucian (2017) grew the Mobs, they did not explicitly set out to separate people and instead used a heuristic for when to form new Mobs. They typically tell everyone that a Mob only is too big when someone in the Mob is not contributing or learning. This means as they grow, they allow the Mob to become larger and larger until there is a key moment in which the team decides to split. They always have one more Mob station available that is needed for the teams to operate independently. This provides an avenue for suggesting that the Mobs split since there is nothing in their way of becoming a separate Mob. This caused us to divide original Mob team into multiple teams to seed culture.

Environment

To accommodate six Mobs, Lucian (2017) needed a new facility. One of the first items that were apparent was wanting everything on wheels, including their 80-inch 4K resolution monitors. This would allow them to rearrange the office at any time. Another change that Lucian (2017) made to their new space was the removal of walls to make more of an open area for them to collaborate more. Lucian (2017) set up pairing stations that laptops could dock at in case anything non-Mob related, e.g., expense reports and other administrative needed to be worked on. Finally, Lucian (2017) wanted to make sure that there was office space for any product owners who would like to be relocated near us to interact with the teams more frequently.

Maintaining Technical Excellence

Attending conferences and a conference budget were two of the highest priorities when Lucian (2017) started as one Mob. Typically, in one year, they would go to two or three Agile Open conferences as well as a major conference like the Agile Alliance. Lucian (2017) found that going together as a team helped build camaraderie and trust. This is a pretty big budget for individual developers to be attending conferences.

Avoid the creation of a tribal knowledge silo

As Lucian (2017) divided the teams and brought on new people, their most experienced Mobbers and developers spread amongst the new teams that formed. They found that if they did not get enough of a knowledge transfer before moving, an experienced person would constantly be asked questions until the Mob built up enough of the common knowledge that they could move forward without that person's knowledge. They called this effect, when a group's collective knowledge was not complete before they lost any single team member, 'creation of a tribal knowledge silo.' They had two solutions to this problem. First was the practice of keeping teams intact and adding the new person before someone else left the team. Second was the incorporation of Gherkin as acceptance tests throughout all our projects. The combination of these two solutions helped them bridge the knowledge gap.

Avoid the creation of internal and external product knowledge silos

When the merge of the Internal and External product teams happened, Lucian (2017) found that there were silos in each team that needed to be distributed across the other teams. Lucian (2017) encouraged the start of movement between Mobs because they want to avoid developing silos within the overall group. They wanted to make sure that there are at least a number of people that can handle any particular task that would otherwise be done by a single developer. Due of the nature of Mob programming, distributing that knowledge was simple and learning the information needed to take care of those tasks was easier as well.

Bi-Annual Goal setting

Traditionally, Lucian (2017) goal setting and evaluation practice was to come up with a few goals for developers to meet and the manager would evaluate them on their performance in these areas. Lucian (2017) decided after retrospection that they wanted to make better use of the biannual goal setting process. They ended up with a process in which each developer helped facilitate another developer in two retrospective formats (see Figure 2.10).

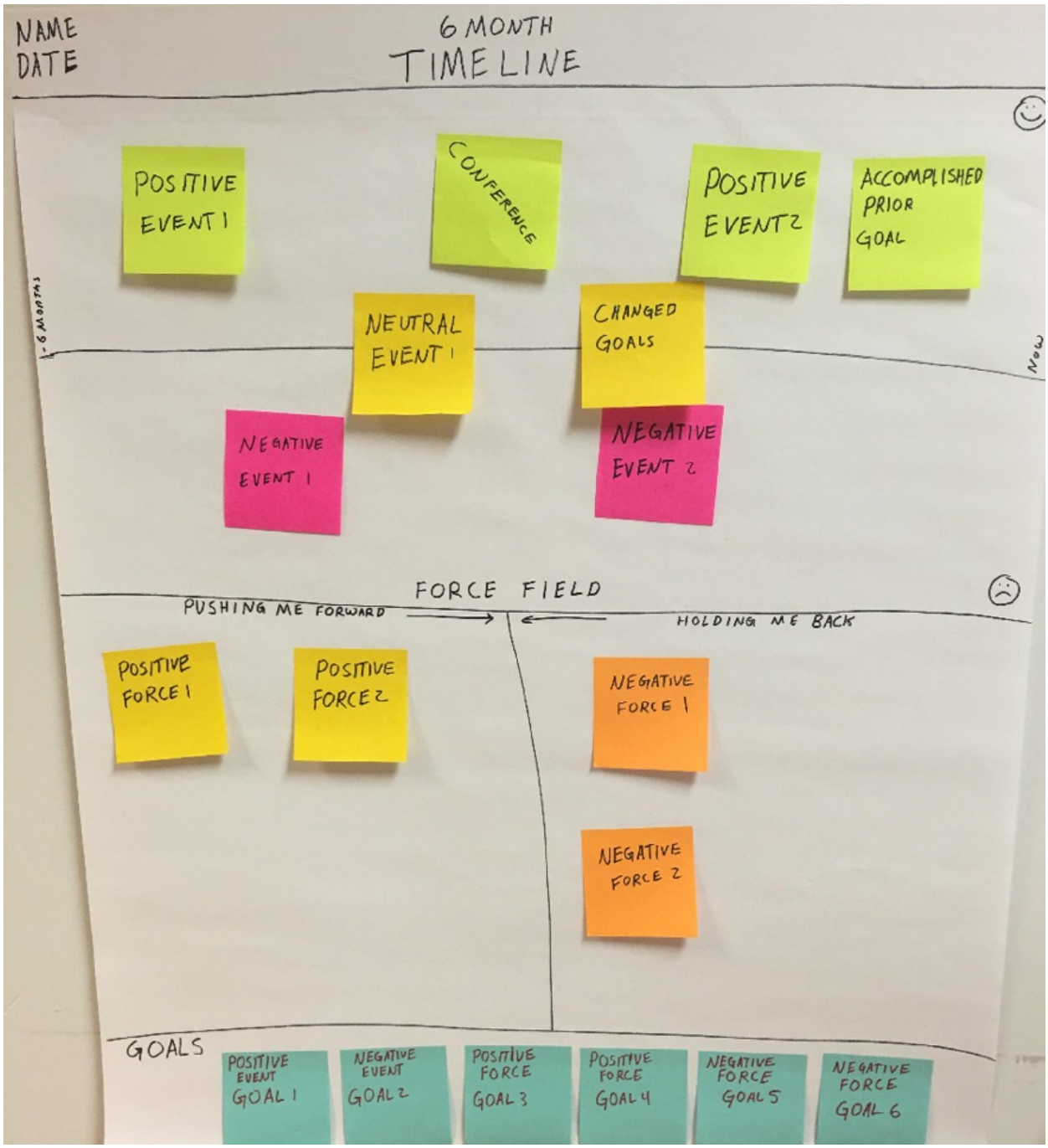


Figure 2.10: *Timeline and Forcefield Retrospective for Bi-Annual Goal Setting. The time line yields insights into events that were positive and negative for the developer and the force field retrospective provides insight into every day behaviors or elements of that person's life that are contributing to or taking away from that person's work experience. from Lucian (2017).*

Figure 2.10 is a timeline retrospective and a force field diagram, to isolate everything that happened to the developer regarding their career in the last six months. The force field diagram

retrospective format is one in which the participant identifies things that are holding them back and moving them forward. The idea is that these are forces pushing a field back and forth. In other words, the timeline yields insights into events that were positive and negative for the developer and the force field retrospective provides insight into everyday behaviors or elements of that person's life that are contributing to or taking away from that person's work experience.

2.3.11 Mob Programming Patterns

The patterns reported by Keeling and Runde (2019) were discovered across both the LendingHome Pittsburgh and IBM Document Ranker teams. Keeling and Runde (2019) chose to focus on patterns that seemed to happen frequently and organized the patterns around three categories: mobbing roles, collaboration patterns, and driving patterns, as summarized in the pattern map shown in Figure 2.11.

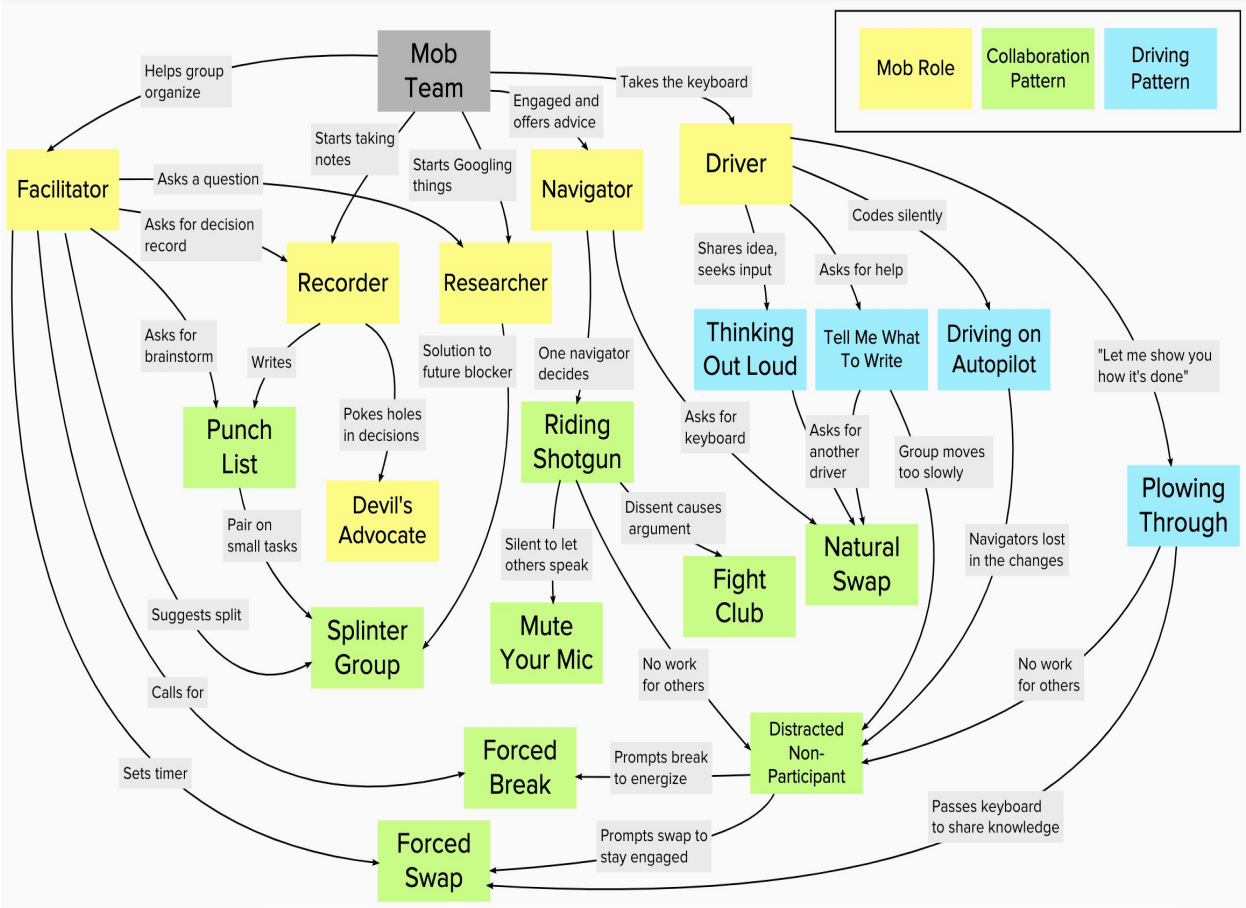


Figure 2.11: *Pattern map summarizing the teams' mob programming patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019).*

Individuals in the mob seem to cooperatively self-organize to fill specific and necessary niches required for the mob to work effectively. Keeling and Runde (2019) categorized this set of patterns as mobbing roles. Table 2.3 summarizes all Mob Programming role patterns harvested by the LendingHome and IBM teams.

Working together as an effective mob takes practice. While trading stories, Keeling and Runde (2019) learned that each team has uncovered different ways of working together. Most of the practices Keeling and Runde (2019) discovered focus on organizing work and improving participation. Keeling and Runde (2019) thinks these patterns emerged in response to an innate desire everyone has to ensure the mob is productive and efficient. Table 2.4 summarizes all Mob Programming collaboration patterns harvested by the LendingHome and IBM teams.

Pattern	Gist
Facilitator	The volunteer helps the group stay focused and to help resolve differences of opinion. Typically the driver does not take this role.
Recorder	Volunteers who capture notes such as context or design decisions on behalf of the mob.
Researcher	Volunteers who seek out information in real-time that is required for the mob to move forward.
Navigator	Direct the driver on what to do. Members of the mob not currently driving are assumed to be a navigator. Navigators can contribute to the mob in many ways.
Driver	The person at the keyboard captures the thoughts of the navigators as everyone works to solve a particular problem.
Devil's Advocate	A navigator who takes a contrarian position to help make the mob's designs stronger.

Table 2.3: *Summary the teams' mob programming roles patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019).*

Pattern	Gist
Punch List	A task list is used by the mob to track work items and select what to work on next. Punch lists can be text-based or graphical.
Splinter Group	One or more members of the mob break away from the main group to complete a routine task. Splinter groups can also investigate alternatives for review by the whole mob.
Ridin' Shotgun	A navigator who solely dictates the mob's work to the exclusion of other navigators.
Mute your mic	A navigator chooses to temporarily remain silent as a navigator to give other navigators a chance to contribute. It is used as a way to kick-start a slow mob or prevent one person from dominating the group.
Fight Club	A situation in which two or more participants fight over the direction of the mob with total disregard for the guiding principles of mutual respect and consideration. Extremely harmful to the mob, considered an anti-pattern.
Natural Swap	A new driver takes the keyboard without prompting by either a member of the mob or timer at a 'natural' break in the work, such as after a test passes or a refactoring step is completed.
Forced Swap	A new driver takes the keyboard after being prompted by either a member of the mob or a timer.
Distracted non-Participant	Navigators who are present in the mob but otherwise do not participate, perhaps distracted by other work.

Table 2.4: *Summary the teams' mob programming collaboration patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019).*

Keeling and Runde (2019) want the navigators to decide the direction of the mob since navigators embody the mob's collective wisdom. Despite this, in Keeling and Runde (2019) experience, the current driver seems to determine a mob's overall effectiveness. Given the driver's outsized influence over the mob, Keeling and Runde (2019) decided to share the different ways their drivers work as driving patterns. Table 2.5 summarizes all Mob Programming driving patterns harvested by the LendingHome and IBM teams.

Pattern	Gist
Thinking Out Loud	The driver articulates their current thinking as they are the prevailing expert in the room or seeing the path forward.
Tell me what to write	Prompt drivers will sometimes use to engage help from navigators, inviting someone from the mob to direct the driver.
Driving on Autopilot	A driver who proceeds without inputs from the rest of the mob.
Plowing Through	A driver who, with the support of the mob, works on the task at hand with the intent of completing it as quickly and painlessly as possible. It combined many times with the thinking-out-loud pattern.

Table 2.5: *Summary the teams' mob programming driving patterns harvested from the LendingHome and IBM teams by Keeling and Runde (2019).*

2.3.12 Strengths and Weaknesses of Mob Programming

Authors	Weaknesses	Strengths
Hohman and Slocum (2002)	Hard to stay focused, especially for navigators. More than two programmers showed the propensity for dispersion of attention, have other conversations unrelated, unless everyone in the room are involved in the practice, the group inevitably becomes uncontrollable.	Improved communication and test coverage. With two projectors, the focal points involve more people there share roles, involving more widely members group. It is suggested two improvements: complete the preparation establish a necessary and coach to oversee indirectly work. The meetings themselves are becoming more productive, regardless of format of one or two projectors.
Wilson (2015)	The effect of dominant personalities within theGroup and appearance of Groupthink.	Dissemination of knowledge was stimulated and the techniqueworked well for critical codes and complex.
Zuill and Meadows (2016)		Strengthens communication presential, Team alignment, Self-Organization, and Collaboration.
Boekhout (2016)	The difference in reaction between the teams indicates that a different approach might be more effective with more senior people.	Not only did they improve their coding skills, they improved in many other aspects, such as their requirements process, deployment procedures, appreciation for focus, and perhaps most important of all, their much higher degree of self-sufficiency. Learning would have been quicker and more directed with a senior as part of the team, but they progressed greatly, even on their own.
Kattan *et al.* (2018a)	There are some tasks that the team does not seem fit for Mob Programming all the times.	The capacity to distribute the knowledge through a streamlined method and working with friends, we observed that increased the learning. Developers approved and enjoyed.

Table 2.6: *Strengths and weaknesses of the Mob Programming*

Table 2.6 summarizes some details founded in literature about the strengths and weaknesses of Mob Programming. Distributed Pair Programming and Mob Programming are topics with a lot of open questions nowadays. Open source projects and home-office work are a kind of remote work, thus Distributed Pair Programming is a relevant problem to be investigated and Mob Programming requires validation through designed empirical research. Pair and Mob Programming seem to increase the learning and satisfaction of the developers compared to solo programming.

2.4 Complexity

There are many software development methods, approaches, practices, and techniques. The literature reviewed objective was looking for better ways to measure the complexity of the work of developers and how complexit's impact can be managed.

When the count of defects takes into account the individual who caused the error, it becomes an evaluation metric, generating the wrong incentives. Deming (2018) says that poor quality is never an individual's responsibility, but rather a management process that allows for the absence of defects to be impossible. Therefore, it is important that an accompanying metric be totally disassociated from any performance evaluation, its purpose should be purely informational.

Padhy *et al.* (2015) carried out a systematic review on metrics and concluded that cyclic complexity, DIT, NOC, AC, EC, and LCOM are well established and valid. Kattan (2015) identifies the same types of complexity that Antinyan *et al.* (2015): internal and external. The internal is the complexity of the function/method itself and the external one is its relation of dependence with other functions/methods.

Antinyan *et al.* (2015) conducted action research at Ericsson and AB Volvo with the participation of researchers from the universities of Gothenburg and Skovde, to identify aspects of code complexity and use the same metrics described here in this section about evaluating the complexity of the code, however, its purpose is to identify the complexity of code to refactor, while the presents a low level of complexity, as they do at Ericsson and AB Volvo.

Nicolaescu *et al.* (2015) argue that the direction of current research should be shifted towards systematizing and evaluating existing results rather than exploring new applicability domains and defining new metric suites.

Mandhan *et al.* (2015) observed a significant level of accuracy in the prediction of the density of defects with the metrics of cyclomatic complexity, DIT, NOC, AC, EC, and LCOM that are part of NASA KC1 PROMISE Repository of empirical software. Kattan (2015) observed in practice the NASA KC1 PROMISE in the measurements of critical and complex systems.

2.4.1 Related to Mob Programming

All the interviewees from Björklund and Fridebo (2020) expressed that mob programming positively impacted the problem-solving aspect and recognized that it was enhanced when working together within a mob:

> "*"I think it's positive with a mob because you always get different angles. When you are alone you sometimes only come up with one idea and you must solve it that way. But when you work with others it brings in other ideas and you get different angles, so it's interesting and you learn things from that too.*"

The development teams that occasionally use Mob Programming expressed the need for the work method when dealing with complex issues that require complicated solutions. An interviewee from Björklund and Fridebo (2020) explained it as:

> " *"That's why we run with mob programming on complex problems precisely because you have four to five people's skills. You can discuss a solution instead of sitting and pondering for three hours. It is faster to arrive at the solution if there are several people. So that part I think works very well, much better."*

According to Aune *et al.* (2018), Mob programming works well for developing fine solutions to critical and complex parts of a system, yet slow at parts that are repetitive and easy. Some of the same benefits and issues from pair programming presented themselves when mob programming in a small team. One of the principal potentials of mob programming appears to be the transfer of knowledge between team members. Programming with others seems to nurture good programming habits, including formatting, design patterns, tool usage, and other small details related to programming languages. Such knowledge may not be acquired from a supervisor or teacher but is rather likely to be passed from student to student. The quality of the product also seems to be fairly good, with defects being corrected continuously. It strengthens the team but can also be draining on energy and mood because of the intense focus required. Mob programming is enjoyable for all involved developers, even though working closely together could bring up the risk of small annoyances.

2.4.2 Related to Pair Programming

One striking result from Melo (2013) was the connection of pair programming with tasks and motivation. When tasks are too easy or too complex, they may influence the motivation of the work. As she has not found references to this phenomenon in the literature, she believes this is a point for further investigation in the use of pair programming for agile team productivity. In addition, the collocation of teams seems not to be enough to reach high productivity in agile teams, but it seems that there is a need to invest in the workspace layout. The following hypothesis can be derived from her results in her opinion: i) the degree of complexity of the task affects motivation to work in pairs, which in turn, affects productivity; ii) productivity varies in function of the workspace layout, even when teams are radically collocated. It is a need for further investigation of these hypotheses in new empirical studies.

Pair Programming is a software development practice in which two programmers work together, side-by-side, on the same module. Studies have shown increased productivity, increased quality, and less risk of knowledge loss due to staff loss. The practice is barely used within NASA's flight software community Dvorak (2009).

2.4.3 Related to complexity management

Vosburgh *et al.* (1984) found that team performance decreases with a higher percentage of complex code. Another research concluded that complex projects require more effort, and coordination, and create a loss of efficiency, which results in lower team performance Boehm *et al.* (2000). However, Fenton and Pfleeger (1998) reported a different result: software complexity might have a positive brunt on the team performance in the context of new development and a negative one in the case of maintenance.

Given that new technologies, capabilities, and requirements are not only inevitable but are, in fact, both mandatory to support the agency's Exploration and Science goals (and are, in fact, enthusiastically embraced by NASA's engineering staff in order to advance their state-of-the-art), it is incumbent on us to find effective ways to identify and characterize software complexity and to find better ways to minimize the impact on cost and schedule of necessary increases in software complexity, while reducing the growth of unnecessary complexity. The potential impact to Mission cost and success as a consequence of high mission complexity can clearly be seen in the Figure 10 (originally generated by Bearden) Dvorak (2009).

How complexit's impact can be managed is from interest of NASA and NSF has open financial research proposals Dvorak (2009). It is an open research problem.

2.5 Information Flow and Knowledge Sharing

To improve the correlation of the research findings related to Source Code Quality Maps, Security Lists, and Hardening Checklists with the related literature about Information Flow and Knowledge Sharing from da Silva and Agusti-Cullell (2008) by integrating both ideas.

In agreement with da Silva and Agusti-Cullell (2008) Maps, can be considered from other perspectives. A specific map is an off-the-shelf Communication Space. Let us think in the following example, illustrated in Figure 2.12 and Figure 2.13.

An agent can draw and annotate a map employing the graphical syntax proposed in the ontology to function as a Communication Space between this agent and a second agent, who going to consult the Map and learn about a determined region through it. Notice that, by consulting the Map, the agent can give up, at least provisionally, direct contact with the region that is represented in the Map.

These two agents illustrated in Figure 2.12, heretofore named respectively, the Cartographer and Reader, interact via the Map, thus, giving up their direct contact with reality on behalf of the artificial information system whose operational interpretation is shared by both.

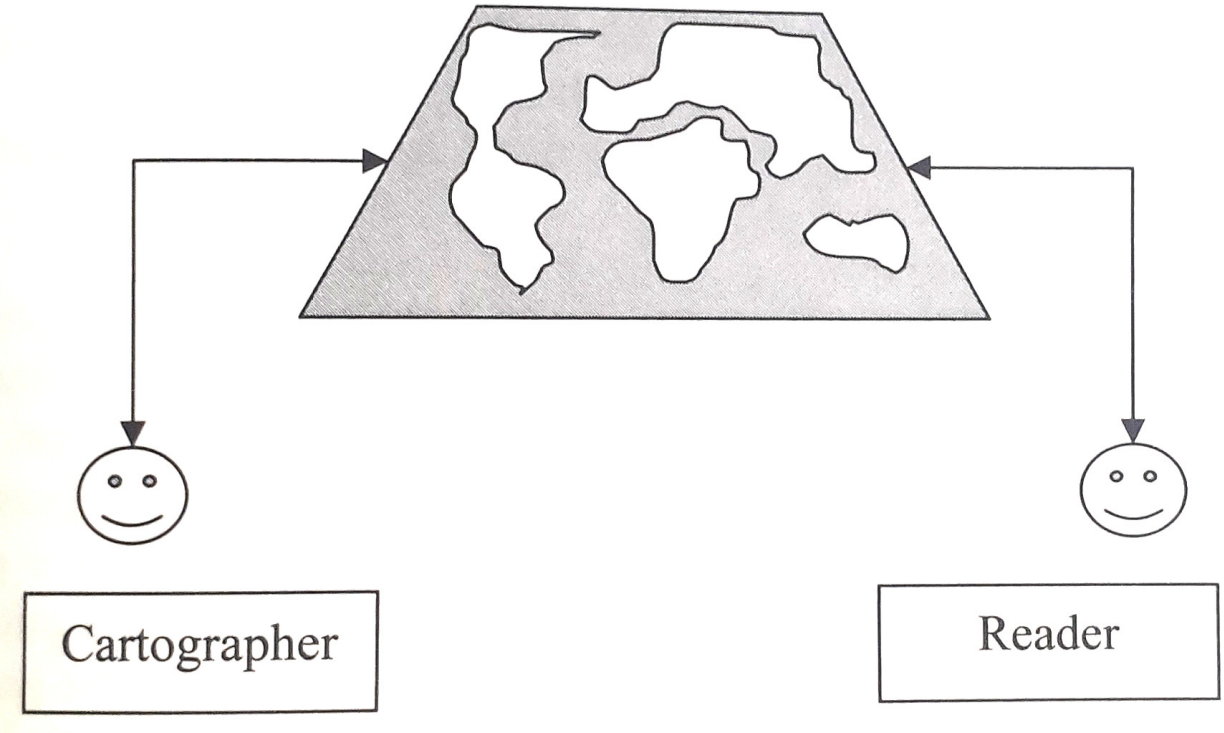


Figure 2.12: *Maps as instantiated ontologies for the interaction between a cartographer and a map reader da Silva and Agusti-Cullell (2008).*

According to da Silva and Agusti-Cullell (2008) Maps, can be considered from **several** perspectives. Let us continue the previous example illustrated in Figure 2.12. But, consider now the case of a Map, e.g., being used by an agent to ask another agent about directions to reach a determined destination. In this case, we have at least three agents engaged in interaction. The agent who is asking for directions, who we call the tourist, the agent who is providing the directions through the map, who we call the local citizen, and the cartographer who drew the map in the first place (Figure 2.13).

The cartographer makes use of an ontology of geographical representations to build a map. This map is then adopted by the tourist and the local citizen so that they can interact to determine the directions to flow. The three agents are engaged in a process of callaborative problem solving, making use of an instance of a preestablished and agreed upon ontology - the map - which is

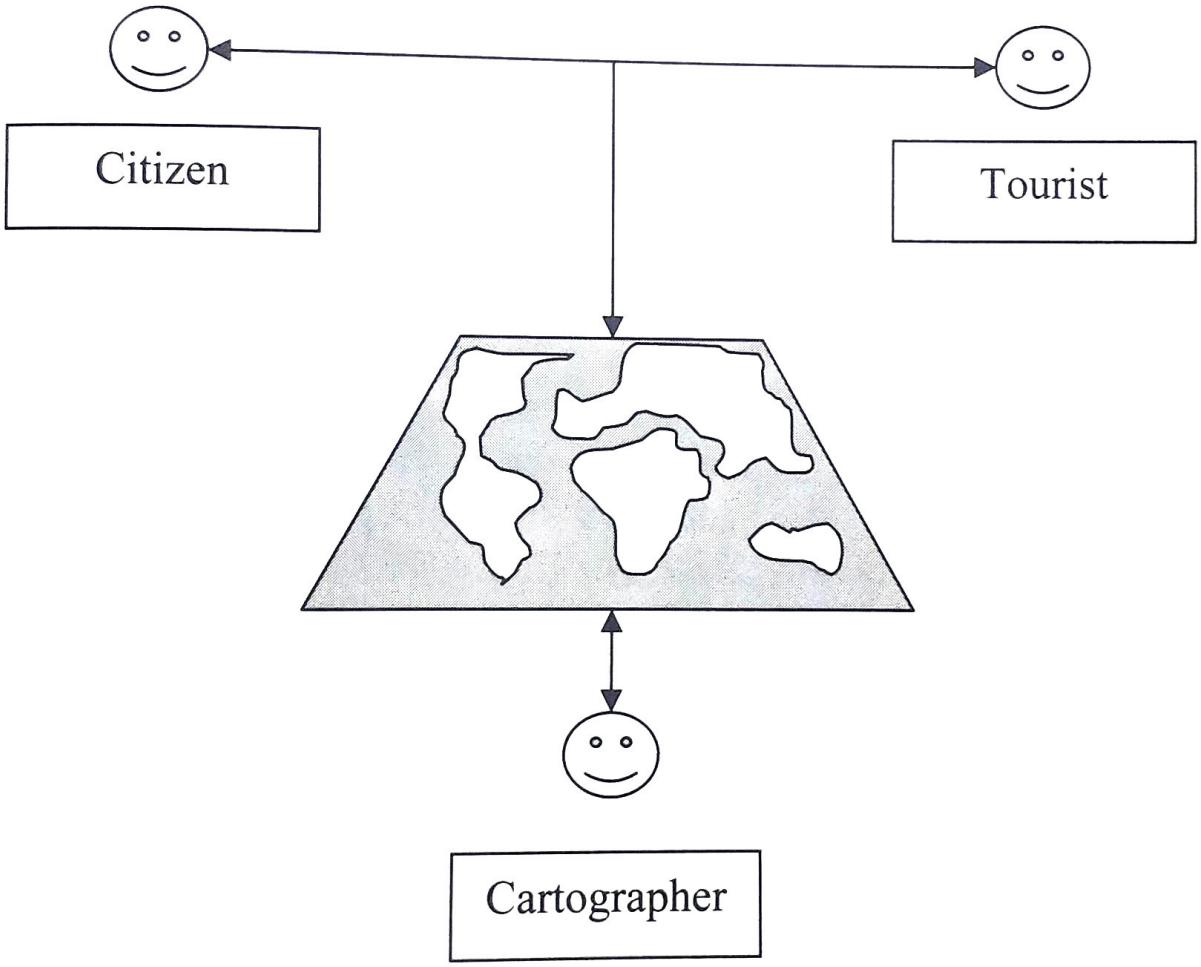


Figure 2.13: *Maps as instantiated ontologies for the interaction among a cartographer, a tourist, and a local citizen da Silva and Agusti-Cullell (2008).*

assumed as a communication space. These three agents have different goals, which are nevertheless synchronized. Evidently, the interaction protocol among these three agents is not symmetrical, in the sense that their participation in the dialogue varies depending on the specific stereotyped roles each of them assumes (cartographer, tourist and local citizen).

In both cases, any analysis of the Communication Space being used by these agents is incomplete if it does not take into account all participating agents in the dialogue and their specific participation in the interaction process based on their perceptions of the reality of themselves and each other, their goals, their selected languages to represent the reality and to interact with each other, and the technological constraints imposed by the available media, tools, and devices that mediate the interaction da Silva and Agusti-Cullell (2008).

2.5.1 Patterns of Interaction

da Silva and Agusti-Cullell (2008) argues the Interaction Patterns are organized by taking into account three dimensions:

- The number of participating agents;
- If the participating agents know each other;
- The interaction occurs in real-time or asynchronous.

2.5.2 The number of participating agents

In accordance with da Silva and Agusti-Cullell (2008), any interaction assumes the participation of at least two agents, fulfilling the roles of sender and receiver. The number of participating agents refers to how many agents assume the role of senders and how many agents assume the role of receiving agents in an interaction.

The possibilities in the case of one and more than one agents for each role. Hence, we have four possibilities:

- 1-1: one sender agent and one receiving agent participate in the interaction. This is the precise case of a dialogue involving exactly two agents;
- 1-N: one sender agent and many receiving agents participate in the interaction. This is the situation, for example, we can observe in a lecture in which the lecturer is the sender agent, and the audience is the group of many receiving agents;
- N-1: many sender agents and one receiving agent participation in the interaction. This situation, which in some sense is the reverse situation of the previous one, can be found, for example, in a group therapy session, in which the patients are all sender agents and the therapist is a receiving agent for them all at once;
- N-N: many sender agents and many receiving agents participate in the interaction. This is the situation that can be observed in a variety of social interactions, such as a company departmental meeting in which all participating agents take turns as sender and receiving agents, or a football match, in which the players also interact by taking turns as sender and receiving agents of information systems.

The class of n-n interactions could be further refined, for example, to consider interactions in which some agents specialize in being senders or receiving agents, thus characterizing more precise patterns of interaction that might suggest more specific desired attributes to be found in an ontology.

2.5.3 If the participating agents know each other

An agent A knows another agent B if A can identify B out of a crowd of agents. In this simple case, we have then two possibilities:

- The agents engaged in an interaction know each other, i.e. they know with whom they are dealing. This is the case, for example, in a company departmental meeting in which the participants all know each other.
- The agents engaged in an interaction do not know each other, i.e. they are interacting anonymously with each other. This is the case, for example, when an author writes a book and an audience of readers later buys and read that book.

2.5.4 The interaction occurs in real-time or asynchronous

As for the interaction being or not in real-time, we have essentially two possibilities:

- Real-time interaction is that sort of interaction that requires the simultaneous attention of all participating agents. This is the case, for example, of a phone call, a company departmental meeting, and a lecture;
- Non-real-time interaction, on the other hand, permits agents to interact asynchronously. This is the case, for example, of electronic mail messages, letters, maps, and books.

According to da Silva and Agusti-Cullell (2008), the combination of these three dimensions gives us sixteen patterns of interaction. Each Pattern of Interaction induces set Patterns for the arrangement of the Communication Spaces, which drive the design and construction of specific ontologies.

2.6 Ontology

Gruber (1995) An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For AI systems, what 'exists' is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.[1]

Ontologies are products designed, built, and sold by specialized companies. Indeed, many companies have been created for this business, some successful. Ontologies have proven to be helpful patterns to build Communication Spaces for particular purpose interactions, which are by design simplified and stereotyped to serve specific shared goals and enable collaborative problem solving based on standardized and simple protocols da Silva and Agusti-Cullell (2008).

2.6.1 Instantiated Ontology

An instantiated ontology is a communication space used for collaborative problem solving da Silva and Agusti-Cullell (2008).

[1] Ontologies are often equated with taxonomic hierarchies of classes, but class definitions, and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world (Enderton, 1972). To specify a conceptualization one needs to state axioms that do constrain the possible interpretations for the defined terms.

Chapter 3

Research Design

This chapter describes the chosen research method. It also discusses the strengths and weaknesses of the research method adopted in this thesis: **Grounded Theory**. In this thesis, we follow the classical version of Grounded Theory. Moreover, this session briefly describes the different versions and presents the justification for the preference for the classical version and how the other versions of Grounded Theory influenced us. Limitations are discussed, as well as the reasons for the choice of Grounded Theory.

Grounded Theory, used as a qualitative research method, studies people and interactions to capture the participants' main concern and how they resolve it. Following is a description of the reasons for choosing the Grounded Theory (GT) research method and the detailing of GT application in this doctoral thesis.

3.1 Introduction

Researchers in the software engineering field argue about the importance of empirical research and the lack of an appropriate research method. It is relevant to increase competence regarding how to apply and combine alternative research methods, tighter links between academia and industry, the development of commonplace research agendas with a focus on empirical methods, and more resources for empirical research Sjoberg *et al.* (2007).

In Software Engineering is very difficult to carry out controlled experiments or make convincing Double Blinding. Moreover, human expertise and subjectivity interfere with outcomes, according to Budgen *et al.* (2006).

In agreement with Kock (2010) and Kock (2004), the Grounded Theory is an appropriate research method for empirical software engineering and solves the problems of another research method called action research. The antidote for the issues of action research is the use of an adaptation of the Grounded Theory research method of Glaser *et al.* (1967). At the core of the grounded theory is a three-step coding process, carried out interactively and according to Carmines and Zeller (1979) is aimed at increasing the reliability of the analysis of large bodies of unstructured research data. The coding process tries to foster objective data analysis and ensure different researchers, regardless of their level of involvement with the environment and subjects studied, will produce the same final data analysis results. This antidote of GT counteracts the weaknesses of all three action research threats to the validity when added alternatives related to the subjectivity involved in the analysis of large bodies of data by fostering intercoder reliability Carmines and Zeller (1979).

Agile methodologies represent a people-centered approach to delivering software. The research on software engineering is much more similar to Social Sciences. This similarity is due to experimental practice, subjectivity, and Blinding Procedures Kitchenham and Charters (2007). Because of the importance of human activities in software development, many of the research methods appropriate to software engineering are rooted in disciplines that study human behavior, both at the individual level (e.g., psychology) and the team and organizational levels (e.g., sociology) Easterbrook *et al.* (2008).

Stol *et al.* (2016) GT has proved an advantageous research approach in several fields, including medical sociology, nursing, education, and management theory. However, GT is a complex method based on an inductive paradigm. It is fundamentally different from the traditional hypothetical deductive research model. Since its inception, GT has provided a helpful methodological approach in numerous disciplines, particularly medical sociology Charmaz (1990), nursing Benoliel (1996), education Opie (2004), and management Kenealy (2008). Moreover, researchers are advocating and training other researchers to use properly Grounded Theory Research Method Nathaniel (2019) Nathaniel (2022).

Whitworth and Biddle (2007) investigates the social processes that contribute to their success. Qualitative grounded theory usage to explore socio-psychological experiences in agile teams, where agile teams prospected as complex adaptive socio-technical systems. Advances in systems theory suggest that human agency changes the nature of a system and how it should be thoughtful. In particular, end goals and positive sources of motivation, such as pride, become crucial points.

Through the Grounded Theory research method, Martin *et al.* (2009) outlines the different roles that were typically on the team, which range from the recognized "Acceptance Tester" function to the less recognized acts of "Political Advisor" and "Super-Secretary."

The reality of the customer role is a critical, complex, and demanding role on agile teams. Despite initial difficulties, customers love agile development and would not do it any other way, but they also encountered many challenges in their day-to-day work. The role has evolved from an individual to a team. As a Grounded Theory reported by Martin *et al.* (2010), the programmers could find it helpful to understand the complexities of the customer's role in the project.

Over the last two decades, there has been an increase in the number of publications exploring the human and social aspects of software engineering through Grounded Theory, all very well-accepted by colleagues of the international community, including several of them awarded by our dears peers in the leading software engineering conferences Hoda *et al.* (2010) Hoda *et al.* (2012) Hoda *et al.* (2013) Hoda (2021) Lindskog and Netz (2021) Hoda (2022).

Grounded Theory is the chosen research method for this research. Institutional support and a successful history of using Grounded Theory for exploring human and social aspects of Agile teams within the department of Computer Science of the Institute of Mathematics and Statistics of the University of Sao Paulo were the primary reasons for selecting Grounded Theory over other applicable methods. Also, in my qualification exam, I was strongly advised to adopt the Grounded Theory as the research method.

Ensuing are the explanations about how the literature review proceeded, first a minor literature review, and after the major one. Both of them it was pursuing the goal of describing it in the most systematic way possible.

3.2 Literature Review

Following the Grounded Theory research method. First, the data was collected and then analyzed. The intention of reviewing the literature after the results is for:

- Protection of the findings from preconceived notions;
- Improve the correlation of the research findings with the related literature by integrating both ideas.

Most research starts with a literature review of some sort Kitchenham *et al.* (2010). However, the researchers must be aware that unless a literature review is thorough and fair, it will have a very little scientific value. Thus, it is the leading rationale for undertaking literature reviews. A literature review synthesizes the existing work in a manner that is fair and seen to be righteous. For example, literature reviews must be undertaken by a predefined search strategy Kitchenham (2012) or at least describe how it did realize. The search strategy must allow the completeness of the search for other researchers to do one appraisal. In particular, researchers performing a literature review must make

every effort to identify and report research that does not support their preferred research hypothesis, as well as identify and report research that supports it Kitchenham and Charters (2007).

Budgen *et al.* (2006) have identified two domains that experience problems with experimental protocols. They are similar to those occurring in software engineering. It investigated these further to assess whether the approaches used to aggregate evidence in these other domains and discover if exists the possibility of adapting them for software engineering. Their experiences from performing systematic literature reviews are positive but reveal infrastructure problems caused by poor indexing of the literature.

Glaser says the researcher can start with a light literature review. Major Literature Review, only as the theory begins to emerge. So, the researcher can conduct an extensive literature review to see how the literature in the field relates to the emerging theory. Glaser argues that researchers use GT to avoid undertaking a literature review very vast before because the emergence of the core category violates the basic premise of GT to emerge a theory based on data without external influences Glaser and Holton (2004).

The rationale behind a minimal literature review before the emergence of the core category is in many ways the same as that behind not starting with a specific research question, namely: avoid clouding the researcher's mind with preconceived ideas and focusing on generating a theory rather than verifying existing ones Glaser (1978).

This research obeys Glaser's recommendation the literature review was kept to a minimum in the beginning-just enough information on related techniques such as pair programming and code review. A deeper understanding of Mob Programming and remote work came from the participants in the early stages of the research.

While extensive literature review in the same substantive area as the research is discouraged early on, reading of substantive areas different from which of the study is considered vital for the researcher to understand how to apply the GT process Glaser (1978). Reading articles and dissertations describing research conducted using GT in other areas, for example, Elliott and Lazenbatt (2005), Urquhart (2007), Nathaniel (2019), Nathaniel (2022), Masood *et al.* (2022), and Hoda (2022), it was helpful.

This research also adheres to the Kitchenham (2012) recommendation to write it systemically. Thus, the chapter of the literature review looking for make be helpful to other researchers Madeyski and Kitchenham (2017).

3.2.1 Protocol of the Literature Reviews

Because the **literature of the Mob Programming is very short** and due to the similarity of both collaborative approaches our goal with this literature review of Pair Programming is to provide an answer to the following research question:

- **"What are the categories founded in the main topics researched about Pair Programming?".**

Fist we discovered what already been researched in this theme and through of this identification of the main topics in the literature of Pair Programming will be done a tertiary review on them together with a literature review of Mob Programming. The objective of these systematic literature reviews is to discover interesting aspects or new insights to guide in the design of this thesis.

Mob Programming is a software development approach where the whole team works on the same thing, at the same time, in the same space, and at the same computer. Mob Programming, as Zuill (2014) describes, is similar to pair programming Beck and Andres (2004), where two people work on the same computer and collaborate on the same code at the same time.

The main difference, compared to pair programming is that the whole team works together as part of the pairing. In addition to software coding, Mob Programming teams work together on almost all tasks that a typical software development team tackles, such as defining stories, designing, testing, deploying software, and collaborating with the customer Zuill (2014) Kattan *et al.* (2018a).

We also reviewed the most relevant literature about Mob Programming. The goal is to provide an answer to the following question with this very short literature:

- **"What already been researched about Mob Programming including the experience reports?".**

Pair Programming

The sources of the search for this paper are IEEE Xplore (ieeexplore.ieee.org), ACM Digital Library (dl.acm.org), SpringerLink (springerlink.com), Elsevier, others websites like Agile Alliance, Google, and Portal for Periodicals of the CAPES (periodicos.capes.gov.br). The search strings used: 'Pair Programming' and 'Programming and Review Simultaneous in Pairs'. The papers were evaluated based on the inclusion criteria: peer-reviewed and a research method applied. The exclusion criteria were: incomplete studies, drafts, slides, abstracts and repeated studies.

We reviewed the literature on Pair Programming looking for reliability, making easier to audit posteriorly and avoid trends in papers selection or interpretation bias made when summarizing them.

We identify categories of main topics about pair programming emerged in this literature review, documenting a useful summary of them, it is based on preliminary finding to help software development teams in the choice of a more appropriate approach for their job context.

After reading all studies, a paper was included if scoped all the inclusion criteria and excluded if had not at least one of the exclusion criteria. To avoid the interpretation bias, the first author performed the complete reading four times, including the interpretations made of all papers. The co-authors of the paper Kattan *et al.* (2018b) also reviewed the interpretations to assure its correction as suggested by the research method used Kattan (2016b) and Kattan (2017a). The results were evaluated based on the inclusion and exclusion criteria described in Table 3.1 according to the guideline of Kitchenham *et al.* (2004)..

Inclusion Criteria	Exclusion Criteria
Peer-reviewed and a research method applied.	Incomplete studies, drafts, slides, abstracts and repeated studies inside literature reviews.

Table 3.1: *Inclusion and exclusion criteria of a paper to be accepted here in this systematic literature review about Pair Programming according to the guideline of Kitchenham et al. (2004).*

Mob Programming

The sources of the search for this paper are IEEE Xplore (ieeexplore.ieee.org), ACM Digital Library (dl.acm.org), SpringerLink (springerlink.com), Elsevier, others websites like Agile Alliance, Google, and Portal for Periodicals of the CAPES (periodicos.capes.gov.br). As Mob Programming is still an emerging software development technique, early adopters publish experiences reports, for this reason, we put one inclusion criteria only to Mob Programming accepting gray literature (not controlled by commercial publishers), although sometimes peer-reviewed, only experience reports, possibly without a rigorous research method.

The search string used in the search was 'Mob Programming' and 'Swarming'. The papers were evaluated based on the inclusion criteria: peer-reviewed and grey literature Schöpfel (2010) only in this case of Mob Programming because how is a new approach has literature available but mostly composed of experience reports. The exclusion criteria were: incomplete studies, drafts, slides, abstracts and repeated studies.

After reading all studies, a paper was included if scoped all the inclusion criteria and excluded if had not at least one of the exclusion criteria. To avoid the interpretation bias, the first author performed the complete reading four times, including the interpretations made of all papers. The others co-authors of the papers Kattan *et al.* (2018a) and Kattan *et al.* (2018b) also reviewed the

interpretations to assure its correction as suggested by the research method used Kattan (2016b) Kattan (2017a).

When the search string used was **'Mob Programming'** return **fourteen** studies, evaluated based on the inclusion and exclusion criteria described in Table 3.2, with emphasis on to exclude abstracts and slides. The **nine** accepted papers passed in all inclusion criteria and the **five** rejected studies did fail in either one of the exclusion criteria.

The results also were evaluated based on the inclusion and exclusion criteria described in Table 3.2 to the search string **'Swarming'** found **five** studies, with an emphasis on to exclude abstracts and slides. The **one** accepted paper passed in all inclusion criteria and the **four** rejected studies did fail in either one of the exclusion criteria.

Inclusion Criteria	Exclusion Criteria
Peer-reviewed, a research method applied and grey literature Schöpfel (2010) only here to Mob Programming.	Incomplete studies, drafts, slides, abstracts.

Table 3.2: *Inclusion and exclusion criteria of a paper to be accepted here in this systematic literature review about Mob Programming according to the guideline of Kitchenham et al. (2004).*

Following is a description of the research perspective, the levels of data abstraction in Grounded Theory, the role of the researcher, and the Grounded Theory with an argumentation justifying our choice and the application of GT in this research.

3.3 Research Perspercitve

An interpretive approach embraces the messy, subjective nature of design studies and emphasizes how it could conduct research of this nature rigorously. Rogers *et al.* (2021) offers an example of the interpretative implementation with rigor criteria to produce a diversity of knowledge contributions. An interpretive perspective widens the focus of a design study's knowledge contribution to include more space for insights separate from the end tool, such as methodology, writing devices, and patterns in social dynamics.

In this research, Grounded Theory was used with an interpretive perspective because the objective was to generate a theory rather than verify existing ones. And pertinent to the conceptual findings outcoming of the study. On the one hand, the strength is the results are modifiable. On the other hand, the weakness is the results are tied to the contexts studied Myers and Avison (2002).

Interpretive studies generally attempt to understand phenomena through the meanings that people assign to them. The main focus of an interpretive perspective is to study a phenomenon by understanding how people make sense of it. An interpretive perspective rejects an objective view of the world and does not attempt to generalize from the sample to a population Myers (2019).

The results derived using this perspective interpretative tie to the context of the study. An interpretive perspective leads to a deep understanding of the phenomenon in a sample context which then could be helpful and inform other contexts Orlikowski and Baroudi (1991).

3.4 Observations

The observations were analyzed and compared to the concepts derived from the interviews. The field observations did not contradict and supported the data provided in dialogues and questionnaires. Thereby strengthening the interview data.

3.5 Levels of data abstraction in Grounded Theory

The rigor of the Grounded Theory method is embodied by the constant comparison method, described in Section 3.8.11. This process is repeated every time a new category is found, changes occur in an existing one, or other properties of a current class are discovered, leading the researcher to revisit previously coded transcripts to see if they have the new property. Figure 3.1 depicts the levels of data abstraction in Grounded Theory. Other codes, concepts, and categories emerged similarly. The emergence of the different kinds is presented in similar diagrams throughout this thesis.

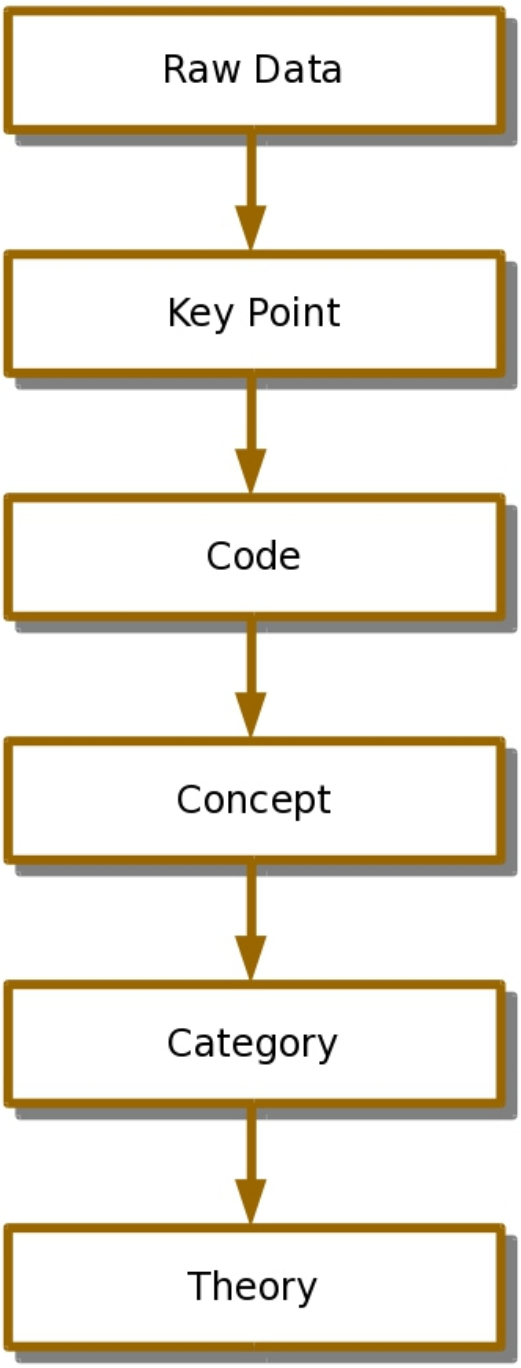


Figure 3.1: *Levels of data abstraction in Grounded Theory*

3.6 Role of the Researcher

Once this research adopted the Grounded Theory research method and chose was using an interpretive approach because of the pursuit of generating a new theory. The role of the researcher is essential in how the phenomenon of Mob Programming is understood and explained. This section arranges background information of the researcher.

I received my Technical degree in data processing from Paula Souza State Center for Technological Education, Sao Paulo, in 1996, my Bachelor's degree in the analysis of systems from Paulista University, Sao Paulo, in 2000, and my Master of Science degree in computer/software engineering from IPT (Institute for Technological Research of the State of Sao Paulo), Sao Paulo City, in 2015. After, in 2016 I started my Ph.D. in computer science at IME-USP (Institute of Mathematics and Statistics of the University of Sao Paulo), Brazil. I am the principal consultant at Herez.com.br. My research interests are pair and mob programming, code review, concurrent engineering, open-source software development, research methods, software engineering, metrics, and computer science in general. I have a video of winning a prize for being the best student in middle school. Also, my Diploma of Honor to Merit from the city of Sao Paulo is available in Appendix I.9.

I learned to love software development and computer in a very similar way to Britain people described in Lean (2016) through BASIC Programming Books Isaaman *et al.* (1982), ISaaman and Tyler (1982), Waters *et al.* (1983), Programming Courses, Games Code Groups Plowman (2017), Computer Clubs, User Groups Stuckey and Swalwell (2014), and Programming in ZX Spectrum McAlpine (2017), Miller (1985). By the way, I also have the original box with the instructions book and others teaching how to program games and BASIC language, as shown in Figure 3.2.



Figure 3.2: *TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research in the original box. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the some books Isaaman et al. (1982), ISaaman and Tyler (1982), Waters et al. (1983).*

I started programming as a child, seven years old, using a TK95 Marecki and Yerzmyey (2019), the Brazilian clone Stachniak (2015) of the ZX Spectrum from Sinclair Research Cline (2019), so many people around the world, learning with it Brooker (2009). And, game programming Stanley and Mawer (2008) was Proof of Coding Skills with no market to sell their games. So many amateur programmers in Czechoslovakia made games to impress their peers using "Didaktik Gama," a clone of the British Sinclair ZX Spectrum, and earn their credit within a community. It also happened in Dutch clubs Švelch (2013), in Poland Wasiak (2014). As well in Portugal Zagalo (2013). Figure 3.3 shows the original back cover box of the ZX Spectrum Brazilian clone, I also put my computer course card with my photo from when a child. Figure 3.4 shows the TK95 Color Computer Keyboard of my first computer, very well conserved. Note the BASIC Language commands in the keys.Figure 3.5 TK95 Color Computer inside the Original Box.

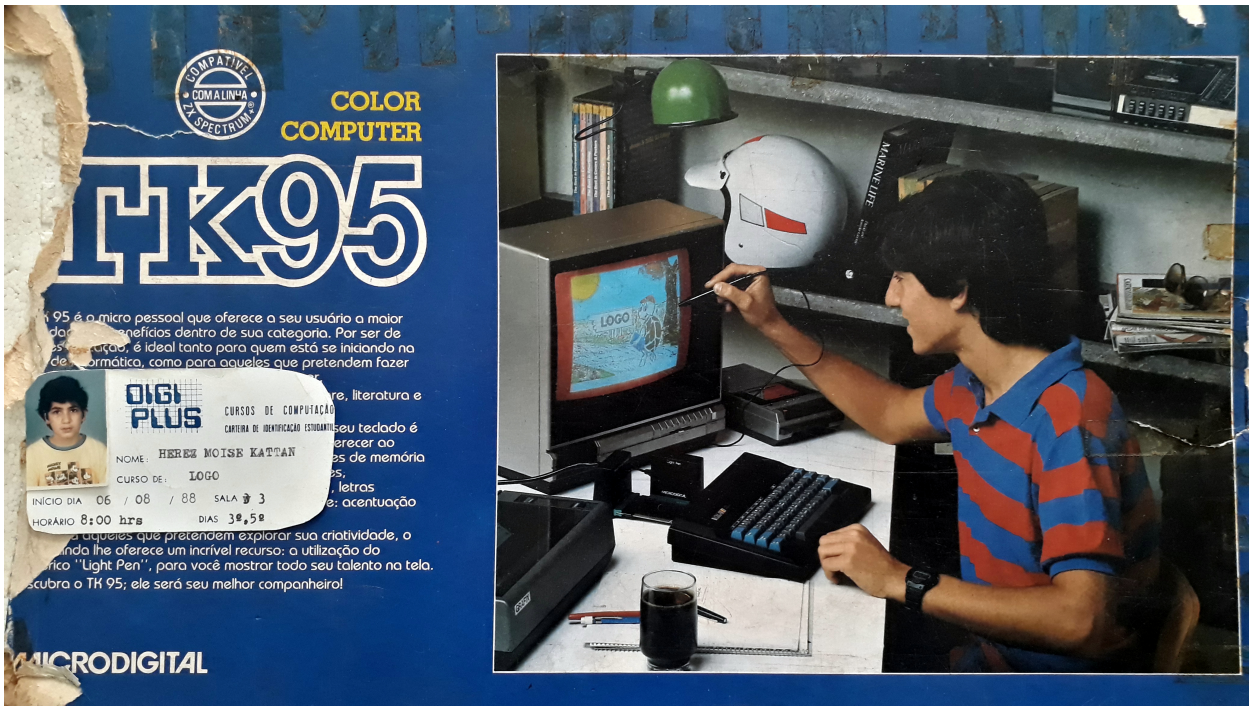


Figure 3.3: *TK95 Color Computer - Original Box - Back View. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the box original of TK 95, a ZX Spectrum Brazilian Clone, working on TV, with three peripherals, the recorder to save the data in cassette tapes, the matrix printer, and the lighting pen to draw on the screen.*



Figure 3.4: *TK95 Color Computer Keyboard. Note the BASIC Language commands are in the keys.*



Figure 3.5: *TK95 Color Computer inside the Original Box. Note a cassette tape on the left superior corner. It was the method to persist your data. On the one hand, it is limited. On the other hand, forty years later, the data are stored with no one sector damaged, and the software runs on the computer without compatibility issues. Moreover, it is years of light ahead of modern computers in energy saving, which is crucial nowadays because of climate change. This is a treasure to be studied.*

The Science Museum Group (2022) from United Kingdown has one ZX Spectrum. Portugal has one entire Museum dedicated to the ZX Spectrum the Museum is called Museum LOAD ZX Spectrum (2022), which was for many Portuguese and foreigners (Spanish, English, Polish, Czechs, Brazilian, and so on), their first personal computer. It preserved the history of this treasure which contributed, in a very decisive way, to the introduction of technology in so many people's lives, and some even built careers in the area, including mine. And, for almost forty years, my first computer it works pretty well due to its well-done internal structure Hardman and Hewson (1983). Figure 3.6 show the good font quality. Figure 3.7 show the back side. More details in Appendix I.8.



Figure 3.6: *Front view of the AC power adapter of the TK95 Color Computer.*



Figure 3.7: *TK95 Color Computer - Back View. The first connector on the right is to plug the TV. The others in the sequence are to connect the earphones, tape recorder, and microphone. Following are the connectors for the joystick and the next one for peripherals, e. g., the lighting pen shown in the back view of the box. The last is to connect the AC power adapter.*

After, as a teenager, in the Data Processing course, as well in the internship assessment, I also had excellent grades available in Appendix I.1. Afterward, as an adult, I was always a good student, my attendance was always 100%, and I never missed a single class, as shown in Appendix I "Background of the Researcher" my background is in software engineering and computer science (Appendix I.2 and Appendix I.3). Additionally, my grades are 83.34% A (Excellent), 16.66% of B (Good), 0% of C (Regular), and 0% of R (Unapproved).

As a professional, I do Pair Programming before the appearance of the agile manifesto. My birthday is on February 11. I remember about lectured training at Paulista avenue on software engineering and the date coincided with emerging of the Agile Manifesto. I was arguing in that class about Pair Programming and teaching similar practices and aspects here in Brazil at the moment of the Agile Manifesto creation. Thus, I could have a bias toward being an agile software engineer. Beyond the experience as an instructor/professor, I also attended so many software development courses (I include some certificates in Appendix I.4).

In 2000 I worked for Microsoft (Appendix I.5), and I had the opportunity to go to the USA to study C# and .NET Platform toward the Brazilian launching road show of these products. It was an excellent experience for me. I learned a lot traveling around Brazil and speaking in courses and mini-courses about C#, Visual Basic, SQL Server, Data Mining, Web Services, XML, SOAP, and the .NET Platform. Also, I love indeed to be hands-on and help customers to build their solutions. Proof of my good performance is a customer of Microsoft Consulting Services satisfied with my job, as shown in the recommendation letter in Appendix I.6. Additionally, I am an author of hands-on technical materials primarily designed for instructor-led training and certification exams related to software development for the official curriculum in the software industry (some documents to prove in Appendix I.7).

Role of the researcher Consistent with the tenets of objectivism, researchers using the classical version of GT should have as few predetermined thoughts as possible and maintain their role as detached observers in agreement with Hall and Callery (2001) as the researcher of this study; I was a detached observer.

Unbiased to reach a worthy consistency in this research and to be the most impartial possible way. I have in person carried out the entire data collection through questionnaires, interviews, observations, and all data analysis processes, with frequent feedback from my supervisors, colleagues, peers, and industry practitioners. Thus, I tried to reduce at most the risk of bias. Also, the full descriptions are in the near sessions.

3.7 Glossary of general Grounded Theory terms

This section aims to help readers understand Grounded Theory terminology. If you have not any difficulty understanding Grounded Theory jargon, go to the next section. In the following sections, the main procedures of the GT research method are described in detail. As well as the application of GT in the subsequent reports. Table 3.3 and Table 3.4 provides a glossary of general GT terms adapted from Hoda *et al.* (2010), and Hoda *et al.* (2012).

Term	Description
Minor Literature Review	The researcher can start off with a light literature review.
Theoretical Sampling	A process that allows the researcher to collect, code, and analyze the data. And then deciding what data to collect next Glaser (1978).
Open Coding	The first step of analysis starts by collating key points from raw data. These are then assigned a code, i.e., a phrase that summarizes the main aspects in 2 or 3 words Georgieva and Allan (2008).
Constant Comparison Method	A process by which codes arising out of each interview are constantly compared against the codes from the same interview, and those from other interviews and observations, producing higher levels of data abstraction, according to Georgieva and Allan (2008) and Glaser (1978).
Memoing	The ongoing process of writing theoretical notes throughout the GT process. Memos capture the conceptual links between categories as the researcher jots their reflections on different kinds.
Core Category	Several categories emerge as a result of data analysis, and the one that can account for most variations in the data and relates meaningfully and readily with other categories is called the core category Glaser (1978).
Selective Coding	Once the core category is established, the researcher ceases open coding and uses selective coding, a procedure where they code for only the core category and those categories that are closely related to the core.

Table 3.3: *Part one of a glossary of general Grounded Theory (GT) terms adapted from Hoda et al. (2010) and Hoda et al. (2012).*

Term	Description
Theoretical Saturation	When further data collection and analysis on a particular category leads to a moment of diminishing results. In other words, no new insight into the category is generated. The category is said to have reached Theoretical Saturation. The researcher can then stop collecting data and coding for that category Glaser (1978).
Major Literature Review	As the theory starts to emerge, the researcher can conduct extensive literature review to see how the literature in the field relates to their emerging theory.
Sorting	Once the researcher has nearly finished data collection and coding is almost saturated, they can begin arranging the theoretical memos on a conceptual level or Sorting. Sorting results in an outline of the theory describing how the different categories relate to the core category Glaser (1978).
Theoretical Coding	Glaser (2005) lists several common structures of theories or theoretical coding families, which can be used as a framework to describe how the categories relate to each other as hypotheses to be integrated into an understanding of the phenomena studied, generating a new theory related to these categories. This is called Theoretical Coding.
Write up	The final step in GT is writing up the theory, which follows the theoretical outline generated as a result of sorting and theoretical coding.

Table 3.4: *Part two of a glossary of general Grounded Theory (GT) terms adapted from Hoda et al. (2010) and Hoda et al. (2012).*

3.8 Grounded Theory

This section describes the research method chosen, which is the Grounded Theory. Since the origins, definition, goal, versions, data analysis techniques, minor literature review, constant comparison method, sorting, memoing, open coding, selective coding, theoretical coding, and write-up.

3.8.1 Roots and Definition

Grounded theory has its roots firmly in the social sciences. It was developed by sociologists for sociologists Goulding (2002). Glaser *et al.* (1967) argue about the use of Grounded Theory (GT) is founded on the premise which the generation of theory at distinct levels is indispensable for a deep understanding of a social phenomena. Glaser (1992) define GT as:

> **"A general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area".**

Glaser *et al.* (1967) GT refers to a method of inductively generating theory from data. GT studies often focus on unstructured text (e.g. interview transcripts, documents, field notes). However, they may also include structured text, diagrams and images, and even quantitative data Glaser (1992).

According with a literature review of Zarif (2012), the grounded theory method developed in significance and gratitude over the years from the seminal work of Barney Glaser and Anselm Strauss Glaser *et al.* (1967). These two sociologists come from diverse setting and their joint work melds primary background in sociology (Glaser (1978); Glaser (1992); Glaser (1998); Glaser *et al.* (1967); Strauss (1987); Strauss *et al.* (1998)). On the one hand, Herbert Blumer, Evert Hughes and Robert Park trained Anselm Strauss in symbolic interaction at the University of Chicago in the school of qualitative research, wherever Strauss was prejudiced by the pragmatist philosophical tradition (chapter of Charmaz in Gubrium *et al.* (2002); Glaser (1998); Strauss *et al.* (1998)).

Additional Barney Glaser was trained in quantitative methodology and qualitative mathematics at Columbia University by Paul F. Lazarsfeld, a trendsetter of quantitative methods (Glaser (1998); Strauss *et al.* (1998)). Glaser was also trained in theory construction by Merton; particularly in theoretical coding, which Merton learned from Talcott Parsons and others (Glaser (1998)). Furthermore, Glaser acquired training in explication of text at the University of Paris (Glaser (1998)). The grouping of the diverse conditions of Strauss and Glaser, whereas functioning jointly during the early 1960s, created the constant comparative method presently recognized as grounded theory (Glaser *et al.* (1967)).

3.8.2 Objective

Glaser (1978) defines the goal of Grounded Theory is:

> **"To generate a theory that accounts for a pattern of behavior that is relevant and problematic for those involved".**

The nature of the ***theory*** generated by the Grounded Theory method is best understood as an explication of the research findings Allan (2006). It has also been described as ***a general pattern of understanding*** by Creswell (2008) and Lisa M. Given (2008).

The distinguishing features of the GT method are a rigorous analysis method powered by constant comparison of data, called the Constant Comparison method, and the practice of frequently recording reflections on data to elicit relationships between them, called Memoing, described later in this chapter.

The originators of Grounded Theory continued to build up the method over the years independently of each other Zarif (2012). Their separated pathways led to what now is recognized as the *Straussian* and *Glaserian* descriptions of the grounded theory method currently there are two school of thought of grounded theory the *Glaserian School* and the *Straussian School* (chapter of Stern in Morse (1994)).

Because I choose to use the ***Classical*** often referred to as a **Glaserian** GT. You could deduce that I do not want the subjectivism of freewheeling flights of imagination. And quite the opposite, I prefer the objectivism of a theory that evolves, modifies, and adapts grounded in the data. Conforming to the literature review of Stol *et al.* (2016) and Kenny and Fourie (2015) found three different kinds of GT, ***Classical***, ***Straussian*** and ***Constructivist*** as the more widely versions used of Grounded Theory. These three versions are following described in the next session.

3.8.3 Grounded Theory Versions or Schools or Traditions

In this thesis, we adopt the ***Classical*** version of Grounded Theory. Moreover, this session briefly describes the different versions and presents the justification for the preference for the ***Classical*** version. Versions, schools, and traditions are synonyms in this context.

Differences between the two originators of Grounded Theory led to the emergence of the first and the second versions of the Grounded Theory method: Glaser's version of GT, often referred to as the *Glasserian* method or *Classical* GT, and *Strauss* version, called *Straussian* GT. This research employs *classical* GT as it is the dominant form of GT used in software engineering research, and due to a larger quantity of resources available Glaser (2016).

Glaser (1992) perspective is well reflected in the fact that he refers to his version of grounded theory as "classic" GT. He strongly disagrees with Strauss and Corbin's version of GT and has argued that Strauss and Corbin's method is not grounded theory, but refers to it as "full conceptual description". Furthermore, Glaser (2002) has called "constructivist" grounded theory a "misnomer".

The Glaserian or Classical Grounded Theory

One of the precious properties of ***classical*** grounded theory is the autonomy it gives the researcher Glaser (2016). The main reason I choose to use the *classical* version is that it offers that independence. Autonomy means total freedom to let the participants' main concern and conceptual variables emerge and continually resolve the main problem. In short, the classical version allows an approach to complete discovery. Correcting existing research conjecture is not the goal of the *classical* Grounded Theory.

Morse *et al.* (2016) argues the focus of the ***classical*** version of the GT is deductive logic. Denzin and Lincoln (2011) argues that ***classical*** version of the GT is based on an objectivist epistemology which assumes that the researcher is separate from what is being studied and aims for unbiased discovery of new knowledge.

The ***classical*** version of the GT is based on a positivist realist ontology Denzin (2008) that assumes an orderly real world exists which can be objectively observed Clarke (2003).

One of the main reasons that I opted for the ***classical*** version of GT is the Philosophical influence of Objectivism, where there exists a single, correct description of reality; the researcher, therefore, discovers grounded theory from data Bryant and Charmaz (2010). This objectivism appear the questions according to Glaser (1978) during analysis:

- What is this data a study of?
- What category or what property of what category does this incident indicate?
- What is happening in the data?

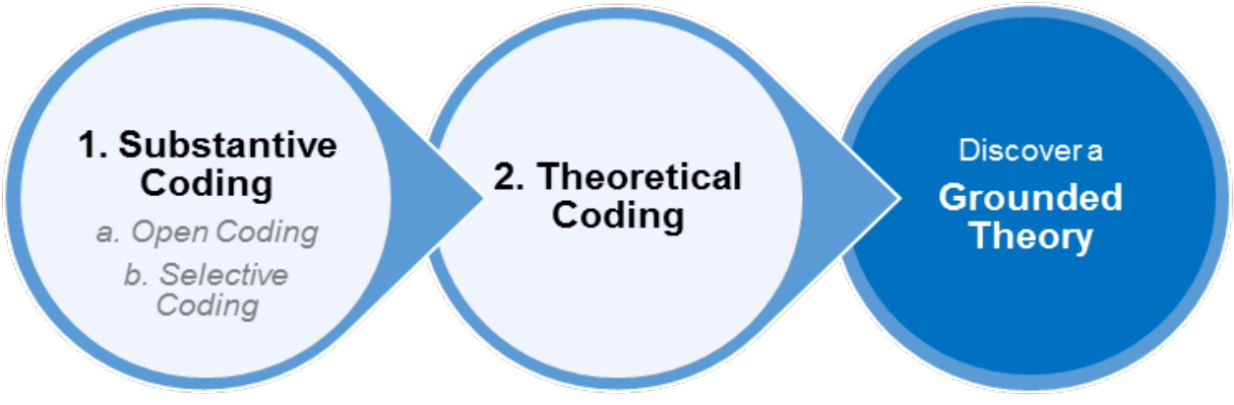


Figure 3.8: *The Coding Procedure of the Classical version of Grounded Theory Holton (2010) and Kenny and Fourie (2015).*

These coding stages showing in the Figure 3.8 are imperative to *Classical* GT as they bind all the concepts of the methodology together and undergird the entire research process from conception to conclusion according to Glaser and Holton (2004) and Holton (2010).

As reported by Onions (2006) in the ***Glaserian*** GT, the theory is grounded in the data Glaser (1978), and in the ***Straussian*** GT, the theoretics is interpreted by an observer of Strauss and Corbin (1994). We call the ***Glaserian*** GT of the ***Classical*** version.

We adopt the ***Classical*** version of Grounded Theory, and in the next session is described the reason for our choice of the ***Glasserian*** version over the ***Straussian***.

The Straussian Grounded Theory

The Basics of Qualitative Research: Grounded Theory Procedures and Techniques. This publication refined and edited specific aspects of the ***Classical*** GT. In particular, Strauss and Corbin (1990) reconfigured the GT coding procedure. They designed a highly systematic and rigorous coding structure to ***create*** (***rather than to discover***) a rigorous theory which closely corresponds to the data Strauss and Corbin (1990),Strauss and Corbin (1994), and Strauss *et al.* (1998). Strauss and Corbin classified four coding stages but qualified that the dividing line between each of the successive phases is somewhat artificial as the researcher constantly moves back and forth between them in consecutive coding sessions Strauss and Corbin (1990). Significantly, this reformation of GT was so distinctive that it became known as ***Straussian*** GT. Their framework is illustrated in the Figure 3.9.

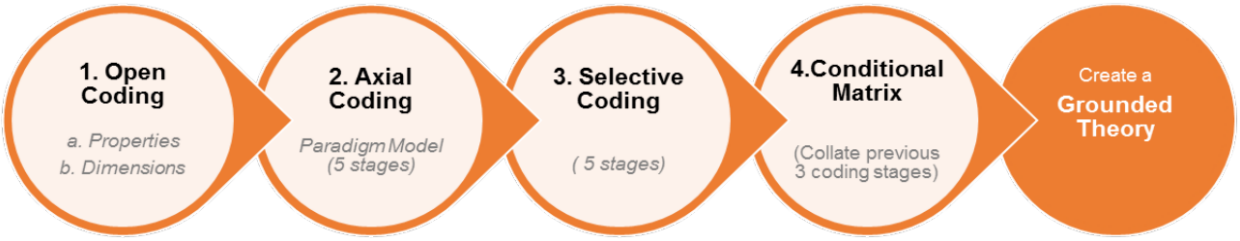


Figure 3.9: *The Coding Procedure of the Straussian Grounded Theory Strauss and Corbin (1990) and Kenny and Fourie (2015).*

GT is growing in popularity as a research method in Computing research areas such as Information Systems and Software Engineering. Despite there are two different GT schools, namely the Glaserian and the Straussian GT schools. A substantial number of research articles tend to ignore the difference and claim that they are using grounded theory. A researcher new to the grounded theory probably finds the two schools look very similar. However, the Straussian school is more prescriptive, and GT newcomers researchers opt to follow this strategy without investigating the Glaserian school. Moreover, the few who try to use a hybrid of the two approaches (not appreciating that the two methods are substantially different) only realize after a significant investment in time that the methods are not reconcilable and that either one or the other should be followed van Niekerk and Roode (2009).

The reason I **not** chose the "Straussian" version of GT is also Philosophical because in the "Straussian" version the influence is Pragmatism and symbolic interactionism: actors engage in a world that requires reflexive interaction. Reality is constructed through interaction and relies on language and communication, according to Chamberlain-Salaun *et al.* (2013), as research design incorporates both methodology and methods, the authors to expose the linkages between the sixteen assumptions and essential grounded theory methods, highlighting the application of the latter in light of the former. So, this pragmatism and symbolic interactionism emerging during the analysis questions about whom, when, where, how, with what consequences, and under what conditions phenomena occur, helps to "discover" important ideas for the theory Strauss and Corbin (1994). "Freewheeling flights of imagination" Charmaz (2014).

The Constructivist Grounded Theory

The Constructivist Grounded Theory Research based on abductive reasoning begins with induction but does not end with it. Instead, the inductive stage leads to formulating hypotheses that are well-established on both: a) First, the researcher's pre-existing theoretical understanding, and b) Second, the emerging research findings and testing them through empirical investigation. Abduction also involves pursuing personal hunches, making highly theoretical interpretations, and making assumptions about any unspoken knowledge Oliver (2012).

Charmaz (2008), a former student of Glaser and Strauss at the University of California, San Francisco forged a radical departure from both Straussian and Classic GT. She presented a third adaptation of GT coding, characterized by a distinctly constructivist philosophy. In stark contrast to Straussian GT, Charmaz resisted a concrete, rule-bound, prescriptive approach to coding, arguing that this stifles and suppresses the researcher's creativity. Instead, she fashioned highly adaptable coding guidelines which endorsed an *imaginative engagement with data*. Charmaz stressed the principle of flexibility in particular, insisting that the analyst must *learn to tolerate ambiguity* and *become receptive to creating emergent categories and strategies*. As illustrated in the Figure 3.10, she proposed a fluid framework, with *at least two stages* to coding.

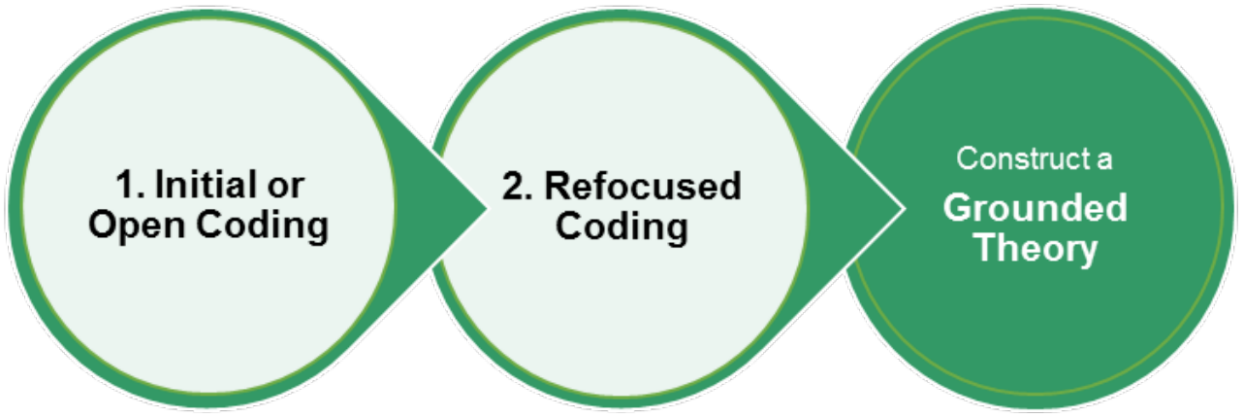


Figure 3.10: *The Coding Procedure of the Constructivist version of Grounded Theory Charmaz (2008) and Kenny and Fourie (2015).*

Charmaz's coding procedure is patently more interpretative, intuitive, and impressionistic than the Classic or Straussian GT Charmaz (2006). Charmaz placed a particularly strong emphasis on in-depth, intensive interviewing to purposely yield an intimate exploration of the meanings that participants attribute to their experiences Charmaz (2006) and Hallberg (2006).

Although these interviews are analysed through the constructivist coding procedure, the analysis rarely culminates into a prognostic or predicative theory presented at the conclusion of the research Hallberg (2006). Instead, a Constructivist GT study typically concludes with the researcher's interpretative understanding (rather than explanation) of the studied social process which is presented in the form of a *story* Hallberg (2006).

Constructivist grounded theorists argue that this narrative approach to GT does not neglect abstraction as it weaves conceptualization into description Charmaz (2006) and Hallberg (2006), particularly as the concluding story encompasses *categories, conditions, conceptual relationships, and consequences* Hallberg (2006).

A type of reasoning that begins with the researcher examining inductive data and observing a surprising or puzzling finding that can not be constructive with conventional theoretical accounts. After scrutinizing these data, the researcher entertains all possible theoretical explanations for the observed data and then forms hypotheses and tests them to confirm or disconfirm each explication until the researcher reaches the most plausible theoretical interpretation of the data Charmaz (2014).

Unlike the *Classical* and *Straussian* schools, Charmaz (2014) assume that neither data nor theories are identified classical either as given gifts in the data or the analysis. Instead, we are part

of the world we study, the raw data we collect, and the reasoning we produce.

Charmaz's reconfiguration of GT was strongly criticized by Glaser (2002). He opposed the constructivist emphasis on descriptive capture, asserting that it *denies and blocks* the *true conceptual nature* of GT. Glaser argued that the unequivocal objective of GT is *conceptualization*, rather than a faithful *description of participants' experiences*. Due to Charmaz's emphasis on the latter, Glaser asserted that Charmaz is *misled* in considering her methodology to be a GT as a more accurate classification would be *Qualitative Data Analysis* Glaser (2002). **I** agree with Glaser that *Qualitative Data Analysis* is a more appropriate classification. Following is **my** rationalization about **my** choice for the *Classical* view and why **I** think adding the word *Interpretative* in the designation of the *Constructivist* version is yet more proper. Thus, **in my view,** the more accurate description of the *Constructivist* arrangement characteristics would be ***Qualitative Data Interpretative Analysis***.

My explanation for the reason **I** also did **not** choose this third *Constructivist* GT starts citing the evaluation criteria of the *Classical* where the generated categories must fit the data, the theory should work, it must be able to explain or predict what will happen. The theory must have relevance to the action of the area, and the theory must be alterable as new data appear Glaser (1978). Moreover, **I** truly believe in a live science objectivist. When new data appears, it needs to adapt and modify the theory. According to Charmaz (2014) the evaluation criteria of the *Constructivist* GT is:

- Credibility (is there sufficient data to merit claims?);
- Originality (do the categories offer new insights?);
- Resonance (does the GT make sense to participants?);
- Usefulness (are useful interpretations?).

Note the *Constructivist* version is very different from the *Classical* version. Observe the *Constructivist* question: "are useful interpretations?", and now observe the *Glaserian* question: "What is happening in the data?." **I** prefer the *Classical* approach because it is traceable in the data. Following are the Uniting and Differentiating Principles of the distinct Grounded Theory versions. They are distinct. However, they are not entirely different. The uniting principles of GT are: Memo Writing, Constant Comparison, Theoretical Sampling, and Substantive versus Formal Theory Kenny and Fourie (2015).

The Uniting and Differentiating Principles of the distinct Grounded Theory versions

The *Classical* GT School also distinguished between *substantive* and *formal theory*. Glaser and Strauss first cautioned a method of generating a GT within a specific arena, such as parental coping with an early diagnosis of their child's hearing loss, yields a limited substantive theory applicable only to this particular field. Whether this theory has broad applicability, for example, parental coping with any trauma, or coping in general, represents a leap from *substantive* (local) to *formal* (all-inclusive) theory and necessitates further study. Glaser and Strauss suggested that *substantive theory* is the bedrock for *formal theory* and advised that the researcher should focus on generating only one or the other during a study Glaser *et al.* (1967). Significantly, *Classic*, *Straussian*, and *Constructivist* GT continues to operate within this framework and maintain this distinction between *substantive* and *formal theory* Kenny and Fourie (2015).

The above precepts (theoretical sampling, saturation, comparative analysis, memos, and substantive versus formal theory) signify quintessential characteristics of GT. These features are deeply embedded within the three GT schools. Thus, Classic, Straussian, and Constructivist GT retain a familial resemblance Kenny and Fourie (2015).

As illustrated in the Figure 3.11, the three schools of GT are distinct entities. However, the researcher does not necessarily have to adopt a pure form of one tradition. Absolutely, within the parameters of consistency, there is the freedom to blur the boundaries between *Classic*, *Straussian*, or *Constructivist* GT. In Kenny and Fourie (2015) experience, they began a research endeavor

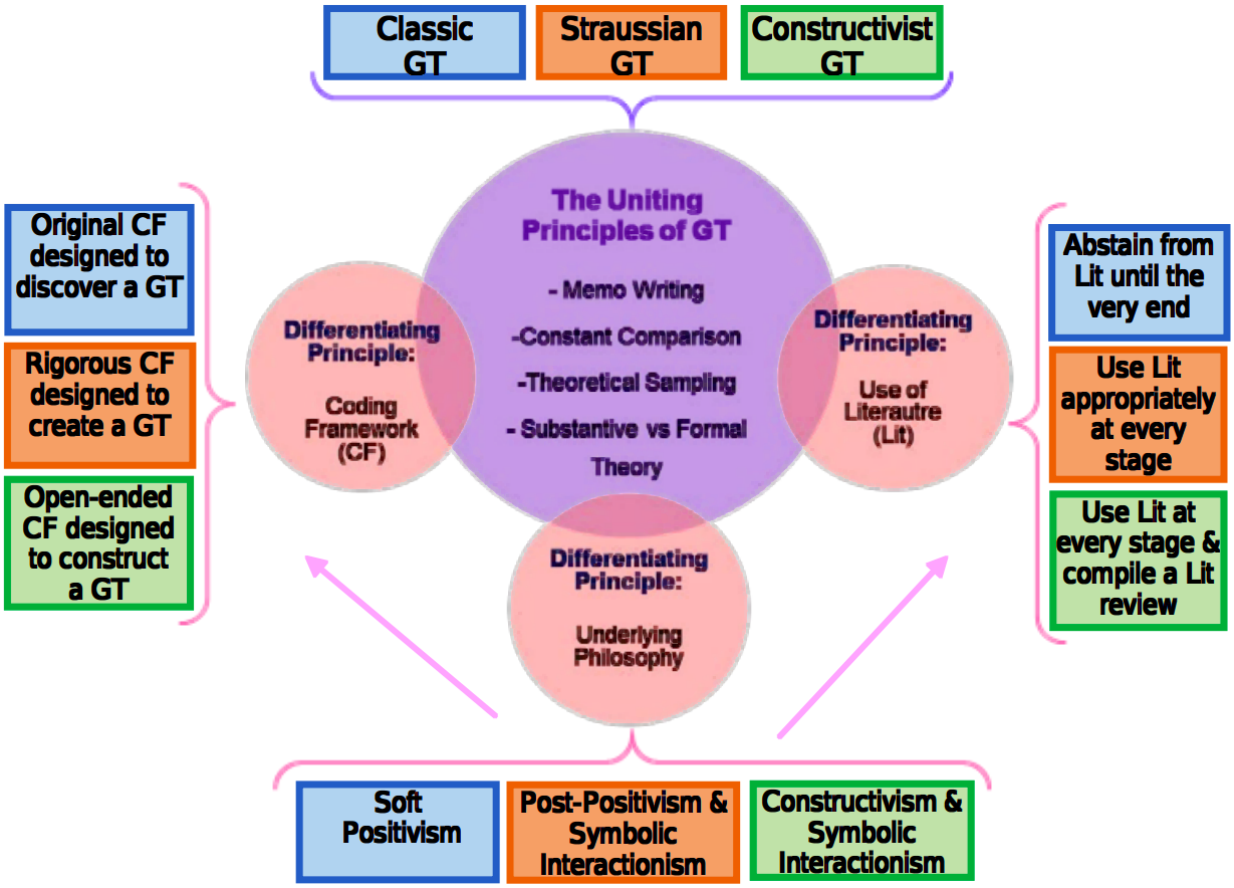


Figure 3.11: *The Uniting and Differentiating Principles of the distinct Grounded Theory versions Kenny and Fourie (2015).*

with a clear identification of *Straussian* GT as the guiding methodology. However, in the data collection and analysis, it became increasingly evident that while the underlining philosophical position was apt, the formulaic structure of *Straussian* GT coding was too rigid for this particular subject of study, which did not naturally fit into Strauss and Corbin's prescribed paradigm model. Consequently, while retaining the philosophical foundation of *Straussian* GT, we relaxed the coding procedure to the extent that it resembled the coding instructions of *Classic* GT. Importantly, this did not violate the integrity of the *Straussian* school, which repeatedly insisted that their guidelines should be employed in a flexible capacity and adapted to best suit the unique dynamics of a study. Thus, while there are clear parameters differentiating the three traditions of GT, and the researcher needs to ensure a consistent approach, there is nevertheless room for creativity and flexibility within the execution of the selected GT methodology.

The GT researcher needs to comprehend the principles that unite and differentiate the three GT schools according to Kenny and Fourie (2015), locate their research within a particular GT version, and defend their rationale for selecting one tradition above the other two. I based on Figure 3.11 from Kenny and Fourie (2015) to help justify my choice of the ***Classical*** version.

3.8.4 Classical School Grounded Theory life cycle

Figure 3.12 presents an overview of the GT method or the GT life cycle from Hoda *et al.* (2012). The diagram captures the main procedures of the Grounded Theory method but does not imply a linear sequence because GT procedures are "cycled and go on simultaneously, sequentially, subsequently, serendipitously" Glaser (1978).

The following sections describe the GT procedures in the order presented in Figure 3.12. Challenges faced in applying the various procedures of the GT method in software engineering research

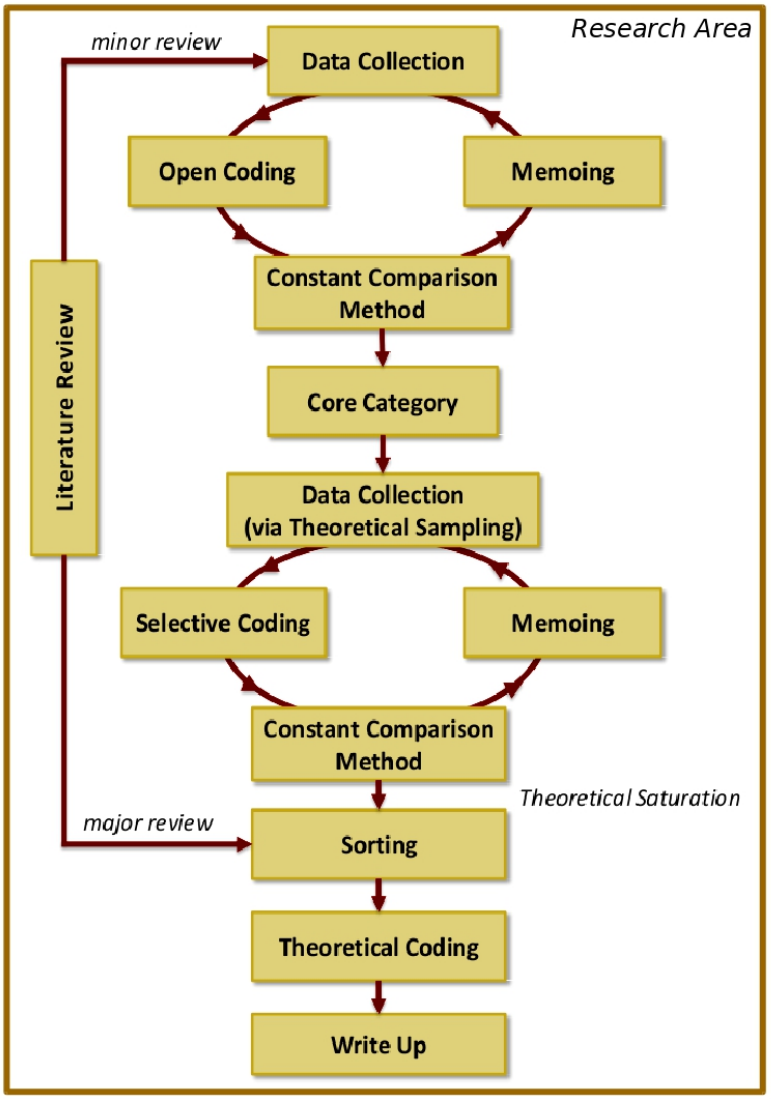


Figure 3.12: *Overview of Grounded Theory research method image adapted from Hoda et al. (2012). The figure was included inside the rectangle Research Area.*

and the strategies found efficient in overcoming them are detailed.

3.8.5 Research Area

Glaser (1978) recommends effectively studying and uncovering the main problems of the participants, and GT recommends refraining from formulating a research problem or a question upfront. The rationale behind this recommendation is:

1. The GT method is meant to generate new theories, and having a preconceived research problem can cause the researcher to be limited in their explorations;

2. The research problem should be the problem of the participants under study and should not be preconceived or forced, instead of it should be allowed to emerge.

The researcher obeys the caution of formulating a research question up front, then only choosing a general area of interest. The plethora of subject areas within Mob Programming makes choosing one a daunting task. This research started by exploring Mob Programming on complex affairs as an area of scientific study. However, the pandemic of COVID-19 unexpectedly brought many challenges

and opportunities, e.g., remote work was mandatory due to sanitary problems. Thus, the area of interest of this research gets wider, including Mob Programing to write high-complexity codes and remote work.

3.8.6 Minor Literature Review

The researcher using the Classical version of the Grounded Theory does a Minor Literature Review upfront to learn some basic terminology used by the peers and be informed and focused on the research area. It is also necessary to conduct an early literature review to find out if the planned study or similar study has been published before. Also, it gives a background to and motivates interest in the particular research area.

3.8.7 Data Collection

This section describes the participant's recruitment process, questionnaire applications, interviews, and observations method. Data collection in GT is a process called Theoretical Sampling and is an ongoing process that helps decide what data to collect next based on the emerging theory. Glaser (1978) define Theoretical Sampling as:

> "Theoretical sampling is the process of data collection for generating theory whereby the analyst jointly collects, codes, and analyzes his data and decides what data to collect next and where to find them to develop his theory as it emerges".

Questionnaire

A list of questions that several people answers and permit a researcher to collect data. Thus, information can emerge from the questionnaire.

A relevant contribution of Bosu and Carver (2013) is an illustration of systematically performing a Software Engineering Questionnaire. Unlike some other more mature fields (e.g., Psychology and Management), most Software Engineering Questionnaire lack a systematic evaluation of reliability and validity, leading to uncertainty in the survey results according to Bosu and Carver (2013). We agree and also believe that to build reliable empirical evidence, Software Engineering questionnaires should be designed systematically and use reliability and validity measures. Thus, we are following the example of Bosu and Carver (2013) to help and guide us on the design of the questionnaire.

The strategy to investigate the hypotheses defined in Section 1.3 and to better understand the effects of the Mob Programming in the software development process is to survey the practitioners of the Mob Programming. The following sub-sections describe the participant selection criteria, survey design process, pilot tests, and data collection methods.

Participant Selection

We invited participants over 18 who are programmers and have developed some software through Mob Programming for at least six months. We had 200 Participants from a diversity of Institutions, Organizations, Companies, and the University of Sao Paulo that adhered to this Study to contribute to the software development community toward a higher knowledge of Mob Programming.

Questionnaire Design

As suggested by Bosu and Carver (2013), we followed well-regarded social and behavioral research methods to the design of our survey according to those described in the books of DeVellis (2011), and Fink and Fielder (2003). Thus, we have two parts to our questionnaire:

1. The first part is about the individual technical knowledge of Programming and Mob Programming.

2. The second part is about the team experience with Mob Programming during this Research.

For the survey, we defined a **Mob-Programmer** as "a person who is using Mob Programming to produce software in a period greater than four months," based on this definition, was designed the survey to determine how **Mob-Programmers** assess their experience using Mob Programming on the following topics:

1. Team members' experience with programming, software development processes, and technologies used by the team using Mob Programming;
2. The room setup. Also, these questions look for a description of the environment in which your team developed software through Mob Programming;
3. Questions related to finding out if a person needs to be absent from the Mob Programming session, the group maintains the same performance;
4. Driver Rotation;
5. Retrospectives and Mini-Retrospectives;
6. Automation of work to avoid downtime;
7. Complex Tasks, large and complex systems;
8. Full involvement all the time can be exhausting;
9. Learning as a team;
10. Scalability and architecture;
11. Learning and sharing technical and problem-domain knowledge;
12. If different techniques have been used together with Mob Programming;
13. Complexity;
14. Satisfaction;
15. Team roles;
16. Collective Intelligence in complex software development;
17. Learning and Mentoring;
18. Remote Work;
19. Continuous Improvement and work in Pandemic times.

We ask for all individuals to complete an online part of the questionnaire about their Programming knowledge and others about their experience with Mob Programming during the job or class. This questionnaire uses qualitative and quantitative research strategies. Data collection and analysis were iterative so that constant comparison of data helped guide future interviews and observations.

In the quantitative part, preferentially open source projects or the source code produced using Mob Programming that is Open Source, available on the internet, and the Programming Language used in the Mob Programming session is compatible with the metrics well-known for code quality.

Similarly, in the qualitative part, all surveys are available in the Appendix A "Questionnaire in Portuguese" and Appendix B "Questionnaire in English" and respecting the respective Brazilian and United States of America laws, all Questionnaires will be available for audit outside of the internet and intranet for five years after the Publication of this Thesis. In addition to questionnaires and interviews, observations were made about the participants' workplaces, such as seating and set-up

of Communicative Space. In an interview and questionnaire, answers are the information that the researchers have to work on. These are all indirect methods, and there is no means to investigate the accuracy. But in the observation method, the information accuracy can be checked by various verifications. However, the questionnaire helped to collect many first data.

Pilot Tests

To ensure that the survey questions were comprehensible and valid with respect to the study constructs, we conducted multiple pilot tests. First, computer science graduate students piloted the survey to identify any difficulties in understanding the questions and to estimate the time required to complete the survey.

On December 8, 2016, we sent a survey invitation to the 14 participants in the LABXP course. The programmers are attendees of LABXP, a regular course offered by the University of Sao Paulo, to graduate and undergraduate Computer Science students. First, the teams watched a lecture about Mob Programming with Alfredo Goldman and Joseph Yoder. They showed one video of Zuill (2012) to the attendees about the time-lapse of a full day of work of a team of Mob Programmers. The course requires a minimum of at least eight hours per week of dedication, and there is a lunch once a week, to allow the students to share experiences.

The conduction of these examples of Mob Programming application in an academic setting with three different teams began in August 8, 2016 and finished on December 12, 2016. All the projects are open source. The first pilot questionnaire was applied to deepen the knowledge about the experience of the fourteen team members.

On December 14, 2016 we sent a reminder email and on December 30, 2016 we sent another reminder email. We closed the Pilot Test survey on January 6, 2017 after the response rate reach 100%. All the answers are available online Kattan (2016a) in the Wiki of CCSL of the IME-USP.

After we analyzed the results of the first pilot test, we fixed some details in the questionnaire. Moreover, we applied this new version again and conducted a second and third pilot test with the students of the LABXP course (starting in the first semester of 2017 and finishing in the second semester of 2018).

Reliability

To reduce uncertainty in survey, according to Litwin (1995), **survey reliability refers to the extent that the survey provides consistent results when replicated in identical conditions**. The three most common methods for assessing reliability are:

1. Test-retest;
2. Alternate-form;
3. **Internal consistency**.

According to Bosu and Carver (2013) **Internal Consistency** is the most appropriate measure of reliability for this kind of Software Engineering questionnaire.

We will use the average inter-item correlation and Cronbach's alpha according to Cronbach (1951) to test **Internal Consistency**. Cronbach's alpha is widely used to calculate the reliability of multiple-item measurements. Table 3.5 summarizes the meaning of the Cronbach's alpha Value indicating our classification of the level of reliability in the Multiple-item Measurements.

While computing alpha is essential to analyze the reliability of a questionnaire, other factors are also important. For example, it is possible to achieve a higher alpha by using more scale items according to Streiner (2003). To combat this problem, we will also calculated the inter-item correlations for the scale items as recommended by Bosu and Carver (2013). For all scales, the mean inter-item correlation will be $> .4$, which is the suggested minimum average interitem correlation according to Clark and Watson (1995). Taken together, the **alpha** and the **inter-item correlation** indicate that the survey instrument is **reliable**.

Cronbach's alpha Value	Reliability of Multiple-item Measurements
alpha >= .9	indicating "**excellent**"
alpha >= .8	indicating "**good**"
alpha >= .7	indicating "**acceptable**"
alpha < .7	indicating "**questionable**" internal consistency

Table 3.5: *The reliability is measured using the Internal Consistency strategy, the evaluation of the probability of the survey to provide consistent results when replicated in identical conditions. The value of the Cronbach's alpha correlating with the respective classification of the level of reliability in the Multiple-item Measurements.*

Validity

According to Litwin (1995), **validity of a survey refers to how well it measures what it is intended to measure**. The four important types of validity measures for a survey are:

1. **Face validity**: is the weakest form of validity. It focuses on analyzing whether "on its face" each item in the questionnaire appear a good operationalization of the construct. We tested face validity by having experts with knowledge of Survey Design and experts with practical expertise in Mob Programming review the survey.

2. **Content validity**: measures the degree to which the scale items represent the domain of the concept under study. We carefully chose appropriate items from prior validated scales to ensure content validity.

3. **Criterion validity**: measures agreement between the survey results and a **gold standard** criteria (e.g. highly regarded prior survey from same domain). In this case, we could not identify another similar survey to act as a gold standard. Therefore, we were **unable to evaluate criterion validity**.

4. **Construct validity**: measures how well a survey instrument performs in practice. It is the most valuable and most difficult validity measure. Construct validity has two forms:

   (a) **Convergent validity**: implies that several different methods for obtaining the same information should produce similar result.

   (b) **Discriminant validity**: implies that measures must not correlate too closely with similar but distinct concepts.

To measure **convergent and divergent validity**, we did perform a principal components analysis with **VARIMAX rotation** using **R**. The only restriction we enforced for the factor solution will be to retain components with **eigenvalue** $>$ **.8**. Eigenvalues appear in many statistical and other computational fields that require transformations of linear systems or are interested in the evolution of software systems from an initial point. Some examples of the applications of eigenvalues include Google's PageRank algorithm, image compression, compounding interest, Markov Chains, and statistical methods such as Principal Component Analysis and Spectral Clustering.

We use the recommendations of Bosu and Carver (2013) as a guide to building this Survey. However, with a unique difference because we prefer to use the statistical software R, instead of the statistical software SPSS used by Bosu and Carver (2013).

Interviews

An interview is a conversation for gathering information. A research interview involves an interviewer, who coordinates the process of the conversation and asks questions, and an interviewee,

who responds to those questions. Interviews can be conducted face-to-face or over the internet. By the way, it is an emerging tool for helping the interviewer carry out the interviewing.

According to Easwaramoorthy and Zarinpoush (2006) interviews are an appropriate method to collect in-depth information on people's opinions, thoughts, experiences, and feelings. Interviews are helpful when the topic of inquiry relates to issues that require complex questioning and considerable probing. Face-to-face interviews are suitable when your target population location permits communication through face-to-face conversations better than they can communicate through video conferencing.

An interview is a qualitative research method that relies on asking questions to collect data. I requested a 20- to 30-minute interview, after I carefully read the answers of the questionnaire. What guided the design choice was Charmaz *et al.* (2012) criticism of the single interview protocol, believing it overused in qualitative research. I did not feel a single 60 or 90 minute interview would be sufficient to gain insight into a learner's social world. Moreover, with a possibility of a second interview request, I had the advantage of determining which experiences were more appropriate for follow up and subsequent theory development. Additionally, doing it after read the answers of the questionnaire helped to guide the interview.

Charmaz (2016) has stated that a researcher's ontological and epistemological stance will dictate the shape and direction of an interview. For example constructionist interviews encourage a definition of situations and events, underlying assumptions, and meanings. Positivist interviews, on the other hand, privilege information regarding events, situations, and behaviours. I attempted to understand events, perceptions, and meanings but at the same time I focused on processes that were occurring, whether of a behavioural or contextual nature.

Once a narrative account was received, I developed a group of clarifying and probing questions. Follow-up questions were intended to gain additional insights into individual experiences and served as a fallback depending on how responsive a participant was in the interview process. The early stage of the interview process commenced with factual questions intended to put the informant at ease. Once preliminaries were done satisfactorily through trial and error, I found it best to lead with an "open-ended" quest or "grand-tour" question before visiting a previously submitted narrative account. Open-ended questions or Grand-tour questions are following described.

- Open-ended questions are free-form survey questions that allow respondents to answer in an open-text format based on their complete knowledge, feeling, and understanding. The response to this question is not limited to a set of options;
- Grand Tour questions are big, sweeping, general questions, asking the interviewee to describe the ground of their experience.

Most interviews contained at least a dozen pre-prepared questions. In some cases, preliminary questions were set aside to allow the participant greater control over the interview process. Later, to expedite the interview preparation process, I highlighted key words in a participant's narrative account, finding this easier to reference during the intensity of the interview space.

Charmaz *et al.* (2012) has provided a list of generic questions and approaches to cocreating and co-constructing knowledge. The questions were useful in assessing the quality of my pre-prepared questions. Preparation for each interview involved a close reading of the previously submitted answers from the questionnaire.

I write many Memos on what appeared to be keywords, actions, and experiences. Following each interview, I reflected and memoed on initial impressions and insights. Also, I overlap with other discussions. An essential aspect of the early interview process was my attempts to understand the Mob Programmer's interactions to solve complex problems. At times, access to information concerning routine operations was challenging; programmers either had differing understandings or appeared uninformed. For example, group size formation decisions were thought strategic, and in another, the respondent hoped group size formation decisions were strategic. Later, I was able to ascertain, from key informants, that group formation relied on a concocting or randomized grouping strategy.

Interviewing

Several interviews were conducted by my WebSite, herez.com.br, using an open-source tool, plugNmeet (Plug-N-Meet - Scalable, Open source web conferencing system) or Rocket.Chat (Open-source communications platform trusted by 12M+ users in over 150 countries), audio recorded, and transcribed. The decision to conduct by my WebSite, the interviews was logistical and assurance of privacy of the participants and other CyberSecurity risks, and it also prevented data leaks.

In the initial phase of the interviewing, I was located in Brazil, at Sao Paulo, whereas the research respondents were located in North America and Brazil. However, some Brazilian cities of the participants are different than mine, and the distance remained a barrier to conducting interviews in person, although arrangements were made for face-to-face interviews with the Sao Paulo city respondents, generally due to taking advantage of observation opportunities, described in the next subsection.

In four cases, research participants resided locally. The audio quality of the recorded interviews was excellent. The audio quality was perfect for the transcription. I also made handwritten notes (Memos) during and following the interviews, which I drew upon. In some cases, the interviews were rescheduled. In other cases, some of the participants, after watching one presentation about Mob Programming and start participating in the research, abandon their participation in the research. The open-source software used in the interviews is described in Table 3.6.

Software	Short Description
Rocket.Chat	The collaboration open-source platform for organizations that handle sensitive business or customer data.
plugNmeet	The plugNmeet is an easy-to-integrate and highly customizable open-source video conference software. The user is classified as a moderator and an attendee. The main difference between the different roles is the capabilities of the meeting rooms. Generally, the moderator has more control over the meeting than the attendees integrated this video conference solution into my Joomla-based WebSite (Herez.com.br) to join a video conference.

Table 3.6: *Open Source Software used in the Interviews.*

They were informed in the Consent document that interviews were recorded and verbal reminders were shared before the start of an interview. While the essential part of the interviews was recorded, not all were fully transcribed, particularly for the second interview. If this happened, only portions of an interview considered relevant to the evolving concepts were transcribed. However, I wrote Memos instead of entire transcriptions sometimes. Although Glaser advises against it, voice recording the interviews helped avoid losing information, and enabled better concentration on the conversations. The interview transcripts served as a good starting point for analysis. Data collection and analysis were iterative so that constant comparison of data helped guide future interviews and the analysis of interviews and observations fed back into the emerging results.

Observations

Data was collected through interviews and was supplemented by observations, over a period of seven years. Face-to-face, semi-structured interviews with Mob Programmers were conducted using open-ended questions as already mention it in previous subsecion.

In addition to interviews, observations were made about the participants' workplaces, such as seating and set-up of Communicative Space, and if alternating with other Agile practices, such as pair programming, mini-retrospectives, pair programming simultaneous style, and structured breaks.

In an interview and questionnaire, answers are the information that the researchers have to work on. These are all indirect methods, and there is no means to investigate the accuracy. But in the observation method, the information accuracy can be checked by various verifications. Thus, the data collected by observation is much more reliable. It used observation techniques to confirm the questionnaires' answers and information in the interviews.

3.8.8 Summary of Data Analysis techniques inside Grounded Theory

Table 3.7, summarizes the data analysis techniques inside GT (the "classical" version as we already mentioned) used by this thesis. The procedure of data analysis called coding in GT. There are two types of coding: Substantive coding and Theoretical coding, Following is the Table 3.7, are summarized the coding mechanisms: Open Coding and Selective Coding, that lead to substantive codes; Memoing and sorting; and Theoretical coding.

Coding Procedures	Short Description
Open Coding	**Fracturing** of the data, line by line coding is recommended by Glaser (1992) to achieve full theoretical coverage, but does not reject coding sentences or paragraphs, or whole documents.
Selective Coding	According to Stol *et al.* (2016), delimiting coding to only those variables that relate to one (or in some cases, several) core variables to establish a parsimonious theory. The core variable guides further data collection.
Theoretical Coding	According to Stol *et al.* (2016), establishing conceptual relations between substantive codes, resulting in the development of hypotheses. Glaser (1978) and Glaser (2005) proposes several **coding families** which are theoretical codes that can be used by researchers, though these must **earn** their way into the emerging theory (e.g. the Six C family in Fig. 3.14).
Memoing	The researcher writes memos (e.g. notes, diagrams, sketches) to elaborate categories as they emerge, describe preliminary properties and relationships between categories, and identify gaps Charmaz (2014). These memos play such an important role in theory generation that stated that, "if the researcher skips this Glaser baldly stage, he is not doing grounded theory" Glaser (1978)
Memo Sorting	also called **Theoretical Sorting** is the continuous process of oscillating between the memos and the emerging theory outline to find a suitable fit for all categories that resulted from the coding according to Glaser (1978) and Strauss *et al.* (1998). Like memoing, Glaser (1992) argues that sorting cannot be skipped.

Table 3.7: *Short descriptions of the data analysis techniques inside GT (the "classical" version) used by this thesis. The procedure of data analysis called coding in GT. There are two types of coding: Substantive coding and Theoretical coding, In this table, are summarized the coding mechanisms: Open Coding and Selective Coding, that lead to substantive codes; Memoing and sorting; and Theoretical coding.*

3.8.9 Definitions of Codes and Coding

The overarching goal of data analysis in the grounded theory methodology is theory development. To achieve this goal, the collected data are evaluated by applying different ways of coding as the core process. **Coding** in grounded theory methodology is a process of conceptual abstraction by assigning general concepts (codes) to singular incidences in the data. After having collected some (not necessarily all) data, the evaluation process may begin Vollstedt and Rezat (2019).

Codes are ideas described in few words that cogently and succinctly explain what is happening in the data. They are the mechanism that allows evolves from raw data to a well-developed theory Chametzky (2016).

According to Hoda *et al.* (2011), the procedure of data analysis, called **coding** in GT, can beginas soon some data is collected. There are two types of coding: Substantive coding and Theoretical coding, that give rise to Substantive and Theoretical codes, respectively. The substantive codes are the categories and properties of the theory which emerges from and conceptually images the substantive area being researched Glaser (2005). In contrast, theoretical codes implicitly conceptualize how the substantive codes will relate to each other as a modeled, interrelated, multivariate set of hypothesis in accounting for resolving the main concern Glaser (2005). Following, we describe the coding mechanisms: Open Coding and Selective Coding, that lead to substantive codes; Constant Comparison Method; Memoing and sorting; Major Literature Review; and Theoretical coding.

3.8.10 Open Coding

Accordingo to Glaser (1978) and Georgieva and Allan (2008) **Open Coding** is the first step of data analysis. We used open coding to analyze the open questions of the questionnaires and the interview transcripts in detail. We began by collating key points from each interview transcript or open question of the questionnaire. Then we will assign a code, a phrase that summaries the key point in 2 or 3 words, to each key point as done in the paper of Grisham and Perry (2005). The codes arising out of each interview were constantly compared against the codes from the same interview, and those from other interviews, open questions of the questionnaires, and observations. This is GT's Constant Comparison method according to Glaser (1992) and Glaser *et al.* (1967) which will be used again to group these codes to produce a higher level of abstraction, called **Concepts** in GT.

Open coding is the part of data analysis that focuses on the conceptualization and categorization of phenomena through an intensive review of the data. In this first step of open coding, the data are broken into smaller parts to be deeply analyzed. This analysis aims to grasp the core idea of each part to develop a code to describe it. Open general concepts can be either developed in vivo, i.e., directly from the data using descriptions that are also derived from or close to the data, or concerning technical literature referring, e.g., to theories from mathematics education, educational psychology, or other relevant areas of study Vollstedt and Rezat (2019).

3.8.11 Constant Comparison Method

The **Constant Comparison Method** will be repeated on the concepts to produce another level of abstraction called a **Category** according to Glaser (1992). Another set of concepts uncovered from the analysis include The end of open coding is marked by the emergence of a **Core Category** Glaser (1992). The **Core Category** is the "main theme" or "main concern or problem" for the participants Glaser (1978). The core is the category that "accounts for a large portion of the variation in a pattern of behaviour" Glaser (1978). There are several criteria for choosing a category as the core. Some of these criteria are: it must be central and related to several other categories and their properties; it must reoccur frequently in the data; it takes the longest to saturate; it relates meaningfully and easily with other categories. We will looking for a category that pass all the criteria for core.

3.8.12 Selective Coding

According to Glaser (1978) once the core category is established, the researcher ceases open coding and moves into **Selective Coding**, a process which involves selectively coding for the core variable by delimiting the coding to "only those variables that relate to the core variable in sufficiently significant ways as to produce a parsimonious theory".

As this research developed categories, the studied core variable undoubtedly emerged. During this phase of analysis, delimit and focus on the recurring issue Stern and Porr (2017) and *related categories* Glaser (1998).

If the categories are not yet sufficiently multivariate, go back to the raw data (interview, field notes, etc.) and code more selectively. It is acceptable, indeed necessary, to re-examine your raw data several times during the analysis; each time you study it, it is possible to do so with a different objective and with another eye. What the researcher has overlooked hitherto will become apparent now Chametzky (2016).

In other words, in the selective code step, we no longer code everything in our data; we code selectively for the core variable to enrich our core variable and its properties. From a practical perspective, there was no difference between open and selective coding was still comparing ideas, incidents, and codes with one another, then writing memos. The difference now was that the focus was on the categories and properties of our core variable.

3.8.13 Memoing

Memos are neglected as a GT procedure. This neglect is partly Glaser's fault to be corrected in the book Glaser (2014) which details the vital aspect of memoing. Memos are a relevant GT procedure because memos are fundamental to the GT generation analysis of grounded theory. The book emphasizes the importance of memos from the start of the GT research to the working paper. Summarizing the main idea is memos help the handling of the plethora of ideas that emerge with no loss hence as GT research progresses.

According to Glaser (1978), the use of **Memoing** helps to theorize the write-up of ideas about codes and their relationships. Moreover, it is vital for recording the relationships between codes. Memos are free-flowing ideas about general concepts and their relationships and are traced to singular incidences in the data but not mixed Glaser (2014). Figure 3.13 illustrates a Memo used in the Mob Programming Team Roles Ontology to the Role of the Driver.

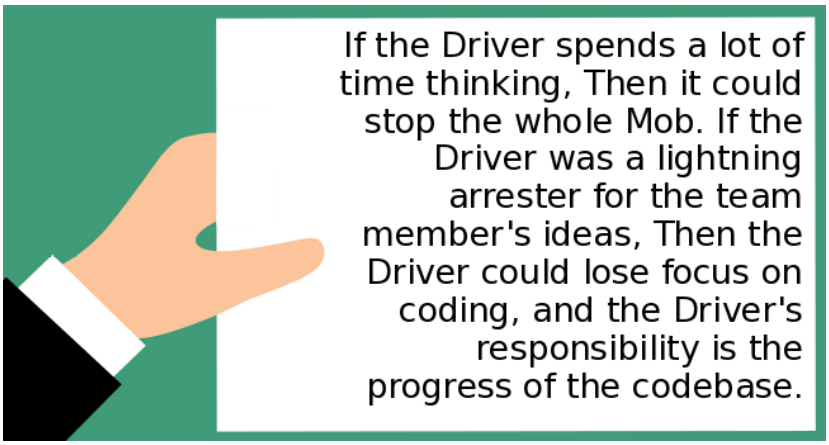


Figure 3.13: *Memo used in the Mob Programming Team Roles Ontology to the Role of the Driver. Photo by mohamedhassan from Freerange Stock. The image was adapted with the text of the Memo.*

Another related challenge is that memoing can become a trivial exercise in tracing where the codes originated, according to Allan (2003). A way to overcome this problem is by avoiding writing about the participants, instead focusing on the codes and concepts, in agreement with Hoda (2011),

the memo in Figure 3.13 does refer to some participant identifiers only as a reminder of their context. The main focus of this memo is the concept of cross-functionality.

Although I agree with Hoda (2011) that it is helpful to record memos electronically on the computer so they can be stored, searched, retrieved, and edited with greater readiness rather than through pen and paper. Separate files for the memos on different topics were created and saved using the topic name for easy recall. This also supported sorting. In my view, writing Memos by hand helps me to reflect in a more cognitive way of sorting.

We wrote Memos when we had ideas about emerging codes and their relationships. Once the data collection was nearly finished and coding was almost saturated, we proceeded to the theoretical Memos toward doing a conceptual sort. Sorting the Memos formed a theoretic outline, showing relationships between concepts.

3.8.14 Sorting

Once data collection is close to finishing and coding is almost saturated, the researcher can begin sorting the theoretical memos to form a theoretician outline. Sorting is an essential step that can't be missed Glaser (1978). The advantage of sorting is that it puts the fractured data back together, according to Glaser (1978). It is to sort ideas, not data. Chronological ordering is not the purpose of sorting. Sorting is done on a conceptual level, resulting in an outline of the theory in terms of how the different categories relate to the core category.

I put together the handwritten Memos with the printouts of all the Memos digitally written in files. They were sorted through their topics. Thus, those related topics were ordered one after the other. An outline of the theory was generated using these topic names in the same order. This outline later formed the draft of this thesis.

The challenge involved in sorting the memos is that while it is easy to group related memos, the ordering of the memos may not be immediately obvious. It takes some shuffling around of memos and thinking out the relationships between the different memo topics, to find an order that makes the most sense. Modeling relationships between the different categories with pen on paper was found to be useful. Once the relationships were established in a diagram (using lines to connect categories), it was easier to spot how the memos (covering different categories and concepts) were related.

3.8.15 Major Literature Review

Concepts are generated from empirical data rather than from existing literature. Similar to a police officer working as a detective who strives to explain what is happening. Thus, the extensive literature review is carried out after the emerging theory and helps to explain the results and see how the existing literature in the field is correlated to the findings.

Classic grounded theory generally prohibits preliminary literature reviews Mccallin (2003), arguing that they might affect emergence. Emergence assumes what is significant in the research area will materialize without presumptions, with the researcher open to what the data represent according to the participants Glaser (1978).

If the researcher is aware of existent concepts found in the literature, these might contaminate the theory, distorting its social reality Glaser *et al.* (1967). To reduce contamination in classic grounded theory, the researcher tends to conduct the 'preliminary' review when generating the theory, to compare it with current literature, and test its application to the research field Deering and Williams (2020) and Christiansen (2011).

Glaser (2005) suggested when the researcher is investigating how the data link to create the codes. Thus, the literature review should be outside the field of study to ensure conceptualization remains original. However, the researcher should be aware that preconceptions may impair the theoretical sensitivity needed to generate an accurate theory Heath and Cowley (2004).

Dunne (2011) draws on his own experience of using grounded theory for his Ph.D. research to explain how extant literature may be used and discusses how the nature of engagement with

existing literature may impact the overall written presentation of a grounded theory study.

This thesis provides a literature review in chapter 2 for the effortless of the reader. However, most of the extensive literature reviews were conducted towards the end of the research, tieing the results into the existing literature. In keeping with the order of the denotative literature review, the research results are followed by a discussion of existing literature, e.g., the results in chapters 4, 5, 6, 7, and 8, initially present the research findings and then discuss them with the existing literature in a discussion section.

3.8.16 Theoretical Coding

According to Glaser (1978) the final step of GT is generating a theory, also known as **Theoretical Coding**. Theoretical coding involves conceptualizing how the categories (and their properties) relate to each other as a hypotheses to be integrated into a theory. Although theoretical codes are not strictly necessary, but "a GT is best when they are used." according to Glaser (2005).

We will Follow Glaser's recommendation, we will use the theoretical coding at the later stages of analysis according to Glaser (1992), rather than being enforced as a coding paradigm from the beginning as advocated by Strauss and Corbin (1990). Glaser (1978) and Glaser (2005) lists several common structures of theories known as theoretical coding families.

Glaser (1978) has provided a list of eighteen coding families or coding paradigms that the researcher may use to analyze provisional concepts and categories. The coding families are intended to assist researchers with their analysis and help identify relationships between Categories and Concepts. The most commonly used coding family is the Glaser's C's, which include causes, contexts, contingencies, consequences, covariances, and finally, conditions. Toward developing complex categories and assisting the researcher in forming a theory or theories about something.

By comparing our data with the theoretical coding families, it emerged that the coding family best adapted to our data was the Six C's coding family, which we described a category in agreement to Kan and Parry (2004) and Picard (2012) about our Six C's model in terms of its Contexts, Conditions, Causes, Consequences, Contingencies, and Covariances.

Additionally, we found a good example related to our field to learn how to do it. This example that we used as a reference is following described.

We used de six C's model from Hoda *et al.* (2011) as an example that guided us while we elaborated our Six C's models. It was helpful, and the title of her paper is "the impact of inadequate customer involvement on self-organizing Agile teams." Moreover, in terms of the Six C's theoretical model for the category Lack of Customer Involvement. Using the Six C's model, Hoda *et al.* (2011) describe:

1. **Contexts**: the ambiance, that is, the context of the Agile development teams in NZ and India;
2. **Conditions**: factors that are prerequisites for the category, Lack of Customer Involvement, to manifest;
3. **Causes**: reasons that cause lack of customer involvement;
4. **Consequences**: outcomes or effects of lack of customer involvement on self-organizing Agile teams;
5. **Contingencies**: moderating factors between causes and consequences, that is, Agile Undercover strategies;
6. **Covariances**: correlations between different categories, that is, how Agile Undercover strategies change when factors that cause Lack of Customer Involvement change. Covariance occurs according to Glaser (1978) and Kan and Parry (2004) when one category changes with the changes in another category.

Figure 3.14 of Hoda *et al.* (2011) is a example of SIX C's model of Hoda *et al.* (2011) guided us to build our Six C's models of this thesis.

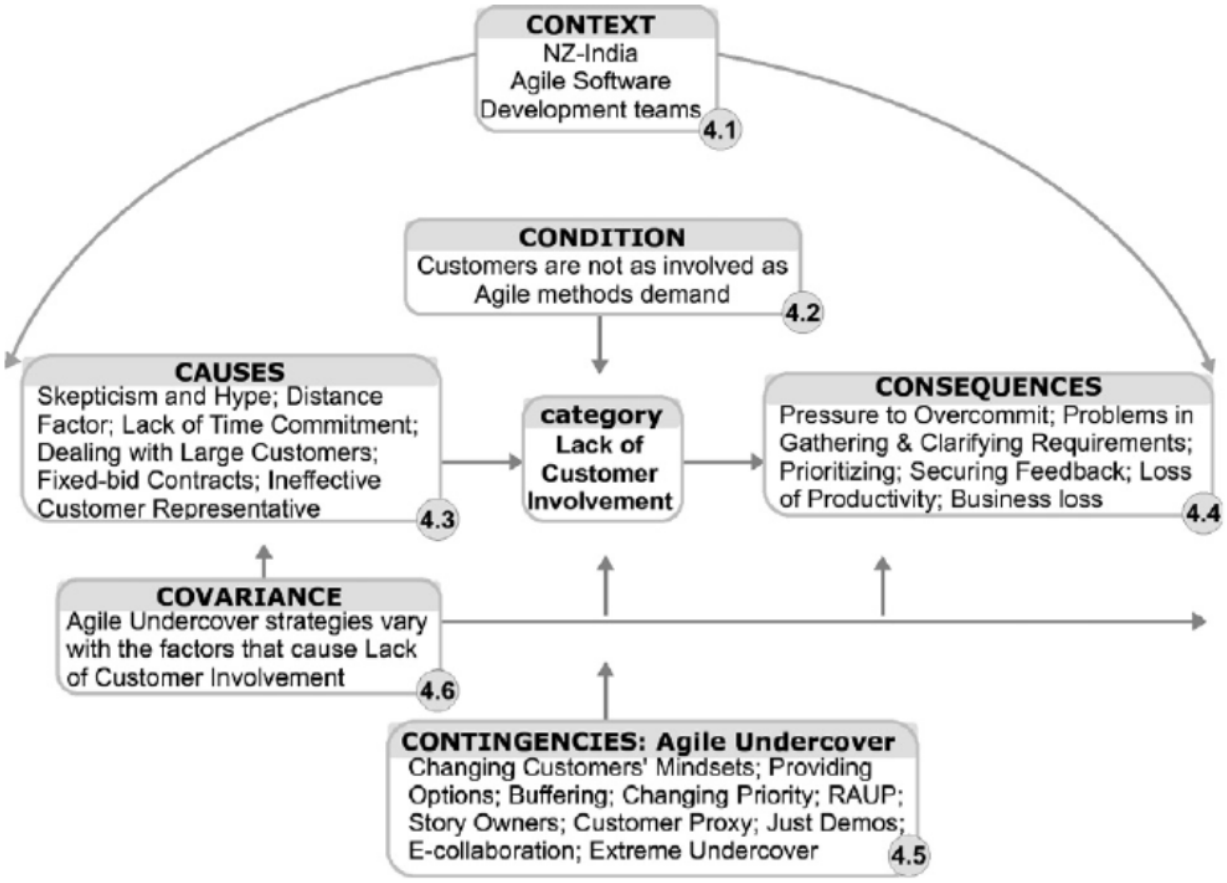


Figure 3.14: *Using Hoda et al. (2011) as an example that guided us in the development of our Six C's model, the title of her paper is "the impact of inadequate customer involvement on self-organizing Agile teams", in terms of the Six C's theoretical model for the category Lack of Customer Involvement. Using the Six C's model, Hoda et al. (2011) describe (1) Contexts: the ambiance, that is, the context of the Agile development teams in NZ and India, (2) Conditions: factors that are prerequisites for the category, Lack of Customer Involvement, to manifest, (3) Causes: reasons that cause lack of customer involvement, (4) Consequences: outcomes or effects of lack of customer involvement on self-organizing Agile teams, (5) Contingencies: moderating factors between causes and consequences, that is, Agile Undercover strategies, and (6) Covariances: correlations between different categories, that is, how Agile Undercover strategies change when factors that cause Lack of Customer Involvement change. The theory of Lack of Customer Involvement depicted using the Six C's model (Context, Condition, Causes, Consequences, Contingencies, and Covariance).*

3.8.17 Evaluating a Grounded Theory

Glaser recommends that a grounded theory should be evaluated on the basis of four criteria: fit, work, relevance, and modifiability Glaser (1992). Following are their descriptions:

- Fit refers to "the ability of the categories and their properties to fit the realities under study in the eyes of the subjects, practitioners and researchers in the area" Glaser (1998). In other words, an emerging theory is said to 'fit' if it explains and fits the experiences of participants as well as different practitioners who were not involved in theory generation Rooddehghan *et al.* (2019).

- Work refers to 'the ability of the theory to explain the major variations in behaviour in the area with respect to the processing of the main concerns of the subjects'.

- Relevance is achieved when the criteria of fit and work are met. Relevance evokes instant 'grab' Glaser (1998).

- Modifiability is a 'quality of the theory to be ready for changes to include variations in emergent properties and categories caused by new data' Glaser (1998).

3.8.18 Limitations

According to Adolph *et al.* (2008) the inherent **limitation** of a GT study is that the resulting theory can only be said to explain the specific contexts explored in the study. Since the codes, concepts, and category emerged directly from the data, which in turn was collected directly from real world, the results are grounded in the context of the data.

Data derived from interviews is prone to **bias** according to Parry (1998). We will build the ability and skill to be able to ask questions that can counter-check the data provided. We also ensured authenticity of the data collected through interviews by supplementing it with observations of workplaces and activities. The data derived from observations we will expect not contradict, but rather supported the interview data, thereby strengthening it.

According to Suddaby (2006), researchers have been criticized for using GT too casually, without clarifying that they have appreciated the intricacies of grounded theory, which is not only quite complex but also based on an inductive paradigm that is entirely different from the traditional hypothetico-deductive model. Grounded theory is new modes of interaction and organization using a methodology that is attentive to issues of interpretation.

Because so many GT articles lack methodological detail, it is difficult for readers to assess whether studies use GT or only reference GT as a methodological rationale, according to Charmaz (2014).

According to Stol *et al.* (2016), to avoid method slurring (among other problems), we are following their four broad recommendations:

1. It is important to study grounded theory before starting;
2. Researchers should describe their implementationof GT;
3. Researchers should avoid **borrowing** rhetoric. If techniques have been borrowed from the GT literature, Researchers should simply state that those techniques have been used without discussing GT. Practices including coding, memoing and constant comparison are all part of the contemporary qualitative data analyst's toolbox. They can exist on their own, independent of their proponents or any particular research method. Bringing in GT clouds the issue;
4. Finally, and related to the previous point, Researchers should not claim to have used GT when they have **not**. Researchers should describe how they analyzed data or generated theory. If using another method it should be named. If a Researcher has developed his/her own method, it should be explained.

Lempert (2007) and Bryant and Charmaz (2007) suggest an issue in some studies related to grounded theory methods. Researchers may report ideas as new that have been developed in relevant works of literature, sometimes by other grounded theorists. Careful analysis of relevant extant literature after developing one's grounded theory can provide cues for raising its theoretical level and indicate which conversations to enter.

3.9 Discussion

The research on software engineering is much more similar to Social Sciences. This similarity is due to experimental practice, subjectivity, and Blinding Procedures Kitchenham *et al.* (2004). The literature review of Dyba and Dingsoyr (2008) investigates the benefits and limitations of agile methods and the strength of evidence about it. The main implication for research is a need for more and better empirical studies of agile software development within a common research agenda herein academia with the software industry. Grounded Theory helps in the systematic analysis of the observational data.

According to Sjoberg and Bergersen (2023), creating and understanding concepts are fundamental in the development and acquisition of knowledge. Its use of concepts to categorize physical and abstract phenomena. Moreover, concepts allow generalizing of particulars and abstract details, making the complexity simpler to manage.

A study based on Grounded Theory is likely, to begin with, a question or even just with the collection of qualitative data. As researchers review the data collected, ideas or concepts become apparent to researchers. These ideas/concepts are said to "emerge" from the data. The researchers tag those ideas with codes that briefly summarize the ideas/concepts. As more data are collected and re-reviewed, codes can be grouped into higher-level concepts and categories. These categories become the basis of a hypothesis or a new theory. Thus, Grounded Theory is quite different from the traditional scientific model of research, where the researcher chooses an existing theoretical framework, develops one or more hypotheses derived from that framework, and only then collects data to assess the validity of the hypotheses. The Grounded Theory analyzes raw data and looks for one auditable Theory to explain the findings using Grounded Theory (GT) techniques as suggested by Allan (2006).

3.10 Conclusion

It presented an extensive justification for the preference for the classical version over the Straussian and Constructivist versions of Grounded Theory. Table 3.8 is summarized the well-known research techniques, methods, and approaches used in this research strategy and design.

Research Technique	Considerations
Literature Review	A manner to identify, evaluate, and interpret related research to a specific phenomenon of interest. Grounded Theory suggests doing it after the empirical part.
Interview	Count on asking questions to collect data. A written outline of important points to be covered reflects consideration for all parties concerned.
Questionnaire	A list of questions that several people answers and permit a researcher to collect data. Thus, information can emerge from the questionnaire. Software Engineering questionnaires should be designed systematically and use reliability and validity measures.
Observation	In an interview and questionnaire, answers are the information that the researchers have to work on. These are all indirect methods, and there is no means to investigate the accuracy. But in the observation method, the information accuracy can be checked by various verifications. Thus, the data collected by observation is much more reliable. It used observation techniques to confirm the questionnaires' answers and information in the interviews.
Grounded Theory	A general methodology of analysis linked with data collection that uses a systematically applied set of methods to generate an inductive theory about a substantive area.

Table 3.8: *Software engineering research techniques, methods, and approaches used in this research and their respective considerations.*

Chapter 4

Mob Programming Team Roles Ontology

This chapter presents the core of our Grounded Theory of Mob Programming Team Roles. The theory explains how software development teams take on informal, implicit, transient, and spontaneous roles; perform balancing acts on a set of integrated practices to become an effective Mob Programming team. The Roles are Driver, Navigator, Researcher, Facilitator, Nose, Quality Analyst, and CyberSecurity Analyst.

This chapter also describes the informal Roles which exist in Mob Programming teams. Members of software development teams, both Agile and non-Agile, also fulfill Organizational Roles on the team. The Roles in Mob Programming teams are informal and implicit. Differently from the Organizational Roles, Mob Programming Team Roles are not designated to the individuals who perform them. Therefore, no problem related to the rotation of these roles.

By explicitly acknowledging this Ontology on Mob Programming Team Roles, the entire team could play to people's strengths and celebrate how their individuality can best contribute to the overall effectiveness and flow of the Mob Programming Session.

Figure 4.1 shows the emergence of a Grounded Theory of Mob Programming Team Roles from underlying categories and concepts. Some of the Mob Programming roles are transient because, unlike organizational roles, they emerge in response to challenges faced by the Mob Programming session and disappear or become dormant as the problems subside. The Mob Programming team roles are spontaneous as opposed to formal organizational roles. Thus, they are intuitively arranged by different team members.

Figure 4.1 also provides an overview of Mob Programming Team Roles that we observed as more stable. The following sections describe each of these Mob Programming Team Roles in detail. The descriptions include selected quotations drawn from the questionnaires or interviews that shed particular light on these categories and are spread across participants and by their roles.

The quotations are presented verbatim from the interview transcripts or the questionnaire answers. When they are with square brackets symbol "[ ]" is used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Ellipsis, a punctuation mark "..." used within quotations to indicate that something has been left out, i.e., to indicate an omission from speech or writing of a word or words that are unnecessary to the specific point or able to be understood from of the context presented, the forfeit of one or more items from a construction to avoid repeating the identical or equivalent items that are in a preceding or following making. While double three full stops preceded and followed by spaces " ... ... " indicate combining two sentences referring to the same context but derived from different parts of the same interview or questionnaire. The quotation between pipes, as known as vertical lines, or vertical slash, "| |" means a translation from Portuguese occurred in the sentence.

4.1 Introduction

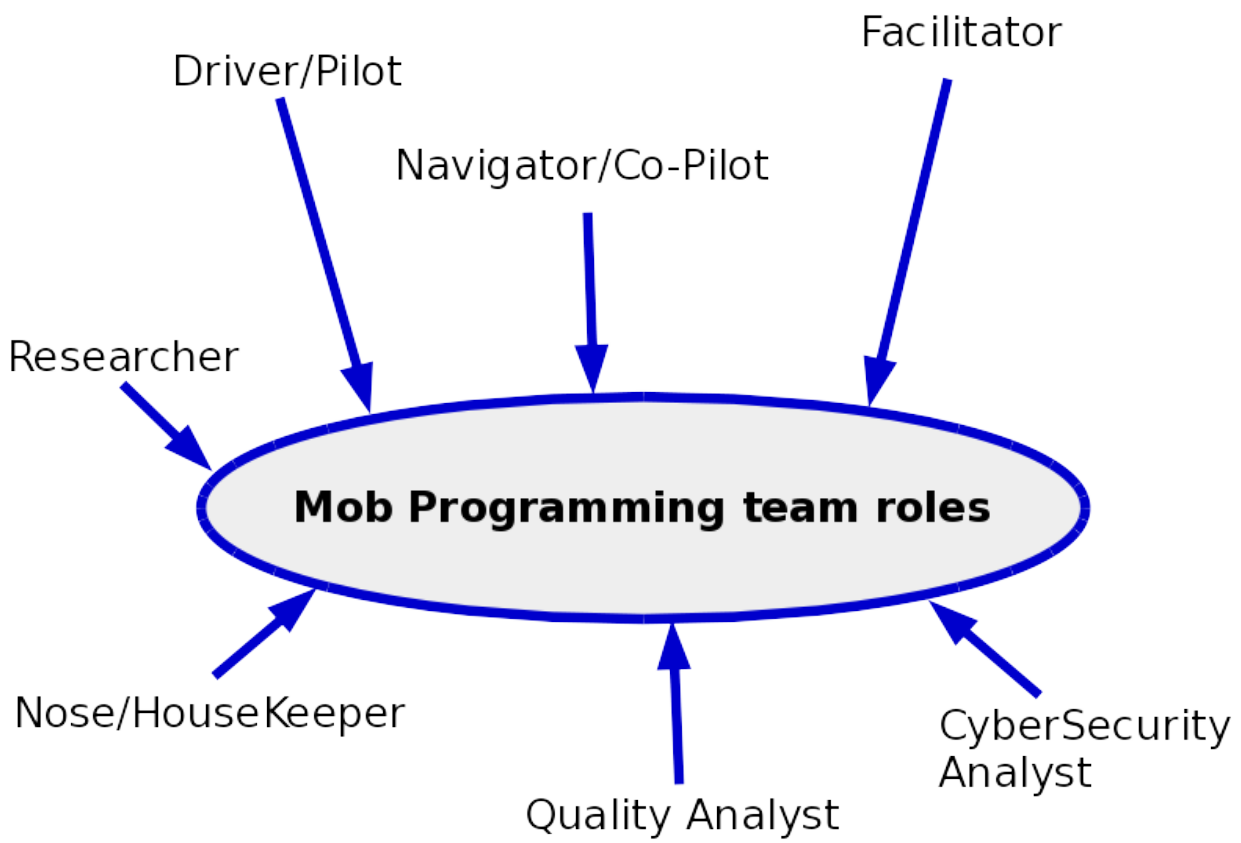


Figure 4.1: *Mob Programming team roles emerged through grounded theory, more specifically in the analysis of underlying categories and concepts.*

The Grounded Theory of Mob Programming Team Roles, beyond the fact of the theory, explains how software development teams take on informal, implicit, transient, and spontaneous roles; perform balancing acts on a set of integrated practices to become an effective Mob Programming team. It was fundamental for understanding how and why Mob Programming works so well in complex tasks and remote work.

The roles described in the following subsections are Driver, Navigator, Researcher, Facilitator, Nose, Quality Analyst, and CyberSecurity Analyst; In the following subsections, we describe the informal roles which exist in Mob Programming teams. The Roles in Mob Programming teams are informal and implicit. Differently from the organizational Roles, Mob Programming Team Roles are not designated to the individuals who perform them. Therefore, no problem related to the rotation of these roles.

Remind that Figure 4.1 also provides an overview of Mob Programming Team Roles that we observed as more stable. Moreover, these Roles appear more than once as an emergent pattern. As shown in the following quotation, not all the teams adopted all the Mob Programming Team Roles listed here.

> "| *We use the roles of pilot, co-pilot, quality guardian, and facilitator.* |" P189

The following sections describe each of these Mob Programming Team Roles in detail. The descriptions include selected quotations drawn from the questionnaires or interviews that shed particular light on these categories and are spread across participants and by their roles.

4.2 Driver or Pilot

Driver or **Pilot** is the person typing on the keyboard, having the input devices, and primarily working on implementing the next step agreed upon with the Navigator or Co-pilot and does not spend time thinking because it could stop the whole team.

The knowledge and skills in the programming language of the driver/pilot also are connected with the way the navigator guides the driver/pilot because the navigator/co-pilot depends on the abilities of the driver/pilot.

A possible analogy related to the Driver/Pilot role is supposed a person of driver/pilot as an intelligent input device powered by artificial intelligence, which the driver/pilot better use and learn the programming language it easier for the navigator to orient.

The driver/pilot's responsibility is especially to code and guarantee the progress of the codebase. The driver/pilot goes on to help to tease out the ideas from the head of the navigator/co-pilot. During the Mob Programming sessions, they learn how to communicate better with each other.

We observed it as a good idea to ignore ideas not coming from the navigator. Otherwise, if the driver/pilot is the lightning arrester for the team members' conversation, it loses focus on coding and spoils your role as the driver/pilot. The driver must be productive and type the source code with attention.

Table 4.1 Mob Programming Team Roles of the Driver or Pilot, with its interactions in real-time at Mob Programming session among the team members. We are detailing the Role of the Drive in a Mob Programming session to help other software developers start using Mob Programming or in the hope that the experienced ones contribute with this ontology to aid our software development community to have a safer and more innovative Communication Space.

Team Role	**Definition**	**Interact with, in real-time at Mob Programming session**
Driver or Pilot	The Driver/Pilot is the person typing on the keyboard, having the input devices, and primarily works on implementing the next step agreed upon with the navigator/co-pilot and does not spend time thinking because it could stop the whole team.	**Navigator or Co-Pilot, Team**

Table 4.1: *Mob Programming Team Roles of the Driver or Pilot, with its interactions in real-time at Mob Programming session among the team members.*

Figure 4.2 illustrates some identified behaviors we consider as productive of the Driver/Pilot, who is the person typing on the keyboard, having the input devices, and primarily working on implementing the next step agreed upon with the Navigator/Co-pilot and does not spend time thinking because it could stop the whole team. Stop the entire Mob is damager than only one person in solo programming. Therefore, the Driver/Pilot's principal responsibility is to guarantee the progress of the codebase.

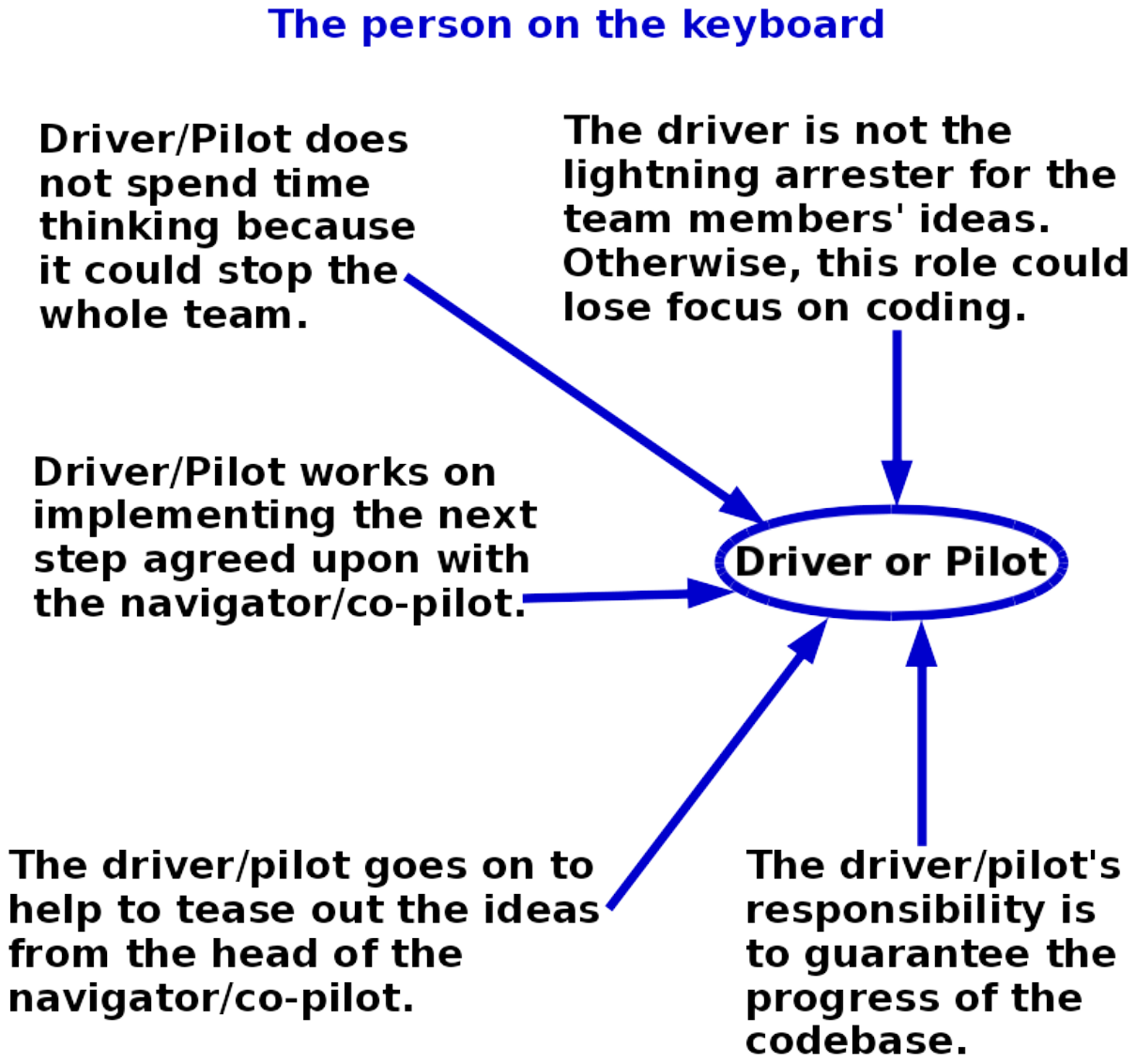


Figure 4.2: *Key points about the Driver/Pilot role emerged through grounded theory when investigating underlying categories and concepts.*

In the section "Discussion" further, in this chapter, we discuss the Patterns of Interaction from da Silva and Agusti-Cullell (2008). Our Table 4.2 is a refinement of the notion of "knowing each other." Thus, producing more refined predicates to create this Ontology. The section "Discussion" described an extremely rough conception of knowledge, perhaps that would satisfy only a policeman trying to identify a suspect: an agent A knows another agent B if A can identify B out of a crowd of agents. Another example would be the doorman of a building trying to recognize if the person is authorized to get into the building or it is not.

Table 4.2 Mob Programming Team Roles of the Driver or Pilot, with its interactions correlation if they know each other and the programming language. It is observed how much more the Driver knows the Navigator, also their style of conducting, which influences communication between them. Thus, they should know each other. How much more they worked together more productive their dialog became. This is an exercise as much as a soccer team training to perform better together.

A consideration about Table 4.2 is the fearless behavior of the Driver if some vulnerability in their knowledge appears ideal to lead with courage because the whole group is looking at the Driver typing and requires emotional intelligence to lead with the situation. Also, it is expected that kind behavior from the rest of the team to together all the team makes a safer and more innovative Communication Space.

Team Role	Team members involved in this interactions knows each other	Team members involved in this interactions knows the programming languase	Interact with, in real-time at Mob Programming session
Driver or Pilot	It is observed how much more the Driver knows the Navigator influences communication between them. Thus, they should know each other.	Driver/Pilot better use the programming language, it easier for the navigator to orient.	**Navigator or Co-pilot, Team**

Table 4.2: *Mob Programming Team Roles of the Driver or Pilot, with its interactions correlation if they know each other and the programming language.*

As shown in the following quotations, the Mob Programming Team Role of the Driver is grounded in the data from questionnaires, interviews, and observations.

"|*Perhaps, it is productive to the Driver to ignore ideas not coming from navigator.*|" P168

"|*The driver's responsibility is to help to tease out the ideas from the navigator's head and especially to digit the code at the keyboard, moving on the codebase forward.*|" P140

"|*A possible analogy related to the Driver/Pilot role is supposed a person of driver/pilot as an intelligent keyboard. It could be an input device powered by artificial intelligence. How much more proficiency using the programming language the person typing will get, it easier for the navigator to orient.*|" P14

"|*There is a correlation between the experience of the driver/pilot including the skills related to the programming language with the way of the navigator guide because the navigator is limited by the abilities of the driver/pilot.*|" P198

4.3 Navigator or Co-Pilot

Figure 4.3 illustrates some identified behaviors we consider as productive of the Navigator/Co-Pilot, who is the lightning arrester for the Mob Programmers' ideas, and primarily working on helping the next step agreed upon with the Driver/Pilot and protect the Driver and help to the Driver not spend time thinking because it could stop the whole team. Stop the entire Mob is damager than only one person in solo programming.

Therefore, the Navigator/Co-Pilot's principal responsibility is to guarantee the strategy of the next implementation helping the Driver in the progress of the codebase and being the lightning arrester for the Mob Programmers' ideas.

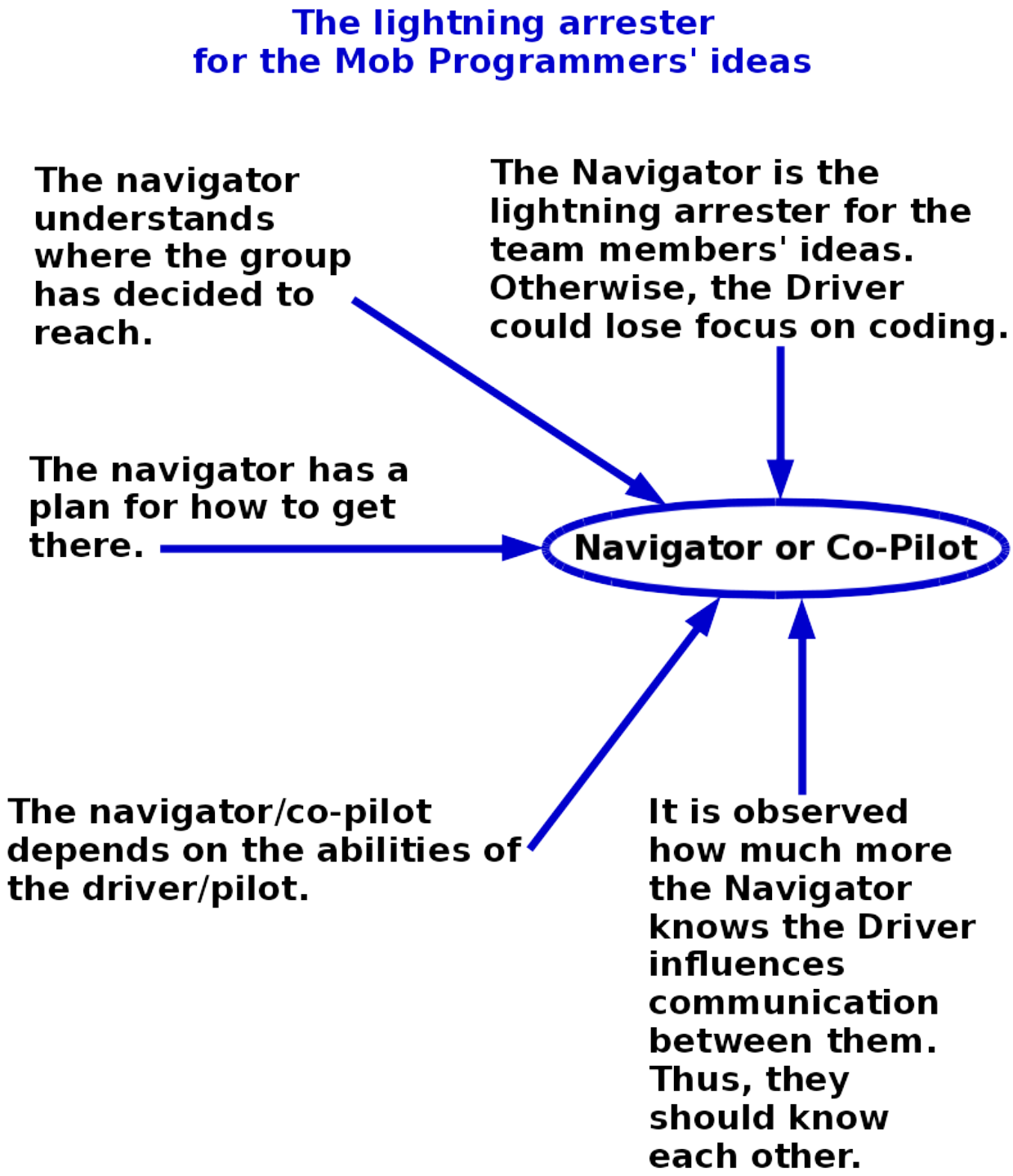


Figure 4.3: *Key points about the Navigator/Co-Pilot role emerged through grounded theory when investigating underlying categories and concepts.*

The Role of the Navigator is the lightning arrester for the Mob Programmers' ideas. The Navigator understands where the group has decided to reach and has a plan for how to get there. In a productive Mob Programming session, the rest of the group will trust them in that role and save any questions or challenges for a later moment, as in illustrated Figure 4.3.

It is important that the navigator is able to communicate intent to the driver, rather than treating them like Alexa, Cortana or Siri and just dictating the code. The driver and navigator need to work together to bring this understanding into the open, and this Productive Dialog between driver and navigator is important for a innovative Communication Space.

Table 4.3 Mob Programming Team Roles of the Navigator or Co-Pilot, with its interactions correlation if they know each other and the programming language. It is observed how much more the Navigator knows the Driver, also their style of typing/coding, which influences communication between them. Thus, they should know each other. How much more they worked together more productive their dialog became. This is an exercise as much as a soccer team training to perform better together.

Team Role	**Definition**	**Interact with, in real-time at Mob Programming session**
Navigator or Co-Pilot	The Navigator/Co-Pilot is the lightning arrester for the Mob Programmers' ideas. The Navigator understands where the group has decided to reach and has a plan for how to get there.	**Driver or Pilot, Team**

Table 4.3: *Mob Programming Team Roles of the Navigator or Co-Pilot, with its interactions in real-time at Mob Programming session among the team members.*

A consideration about Table 4.4 is expected kind behavior from the Navigator and the rest of the team to together all the team make a safer and more innovative Communication Space. In this specific case, the Driver demonstrates a vulnerability. Also, it is expected the fearless behavior of the Driver, if some vulnerability in their knowledge appears ideal to lead with courage because the whole group is looking at the Driver typing and requires emotional intelligence to lead with the situation.

Team Role	**Team members involved in this interactions knows each other**	**Team members involved in this interactions knows the programming languase**	**Interact with, in real-time at Mob Programming session**
Navigator or Co-Pilot	It is observed how much more the Navigator knows the Driver influences communication between them. Thus, they should know each other.	Navigator/Co-Pilot better know the programming language, it easier for the navigator to orient.	**Driver or Pilot, Team**

Table 4.4: *Mob Programming Team Roles of the Navigator or Co-Pilot, with its interactions correlation if they know each other and the programming language.*

As shown in the following quotation, the Mob Programming Team Role of the Driver is grounded in the data from questionnaires, interviews, and observations.

> "| *... Everyone was important in conducting the Mob Programming session so that it was possible to reach the goal at the end of the day because the pilot and co-pilots were responsible for coding and proposing solutions. Without the people accountable for the coding, it would not be possible to achieve the objective, which was system functionality. ...* |" P189

4.4 Facilitator or Mentor

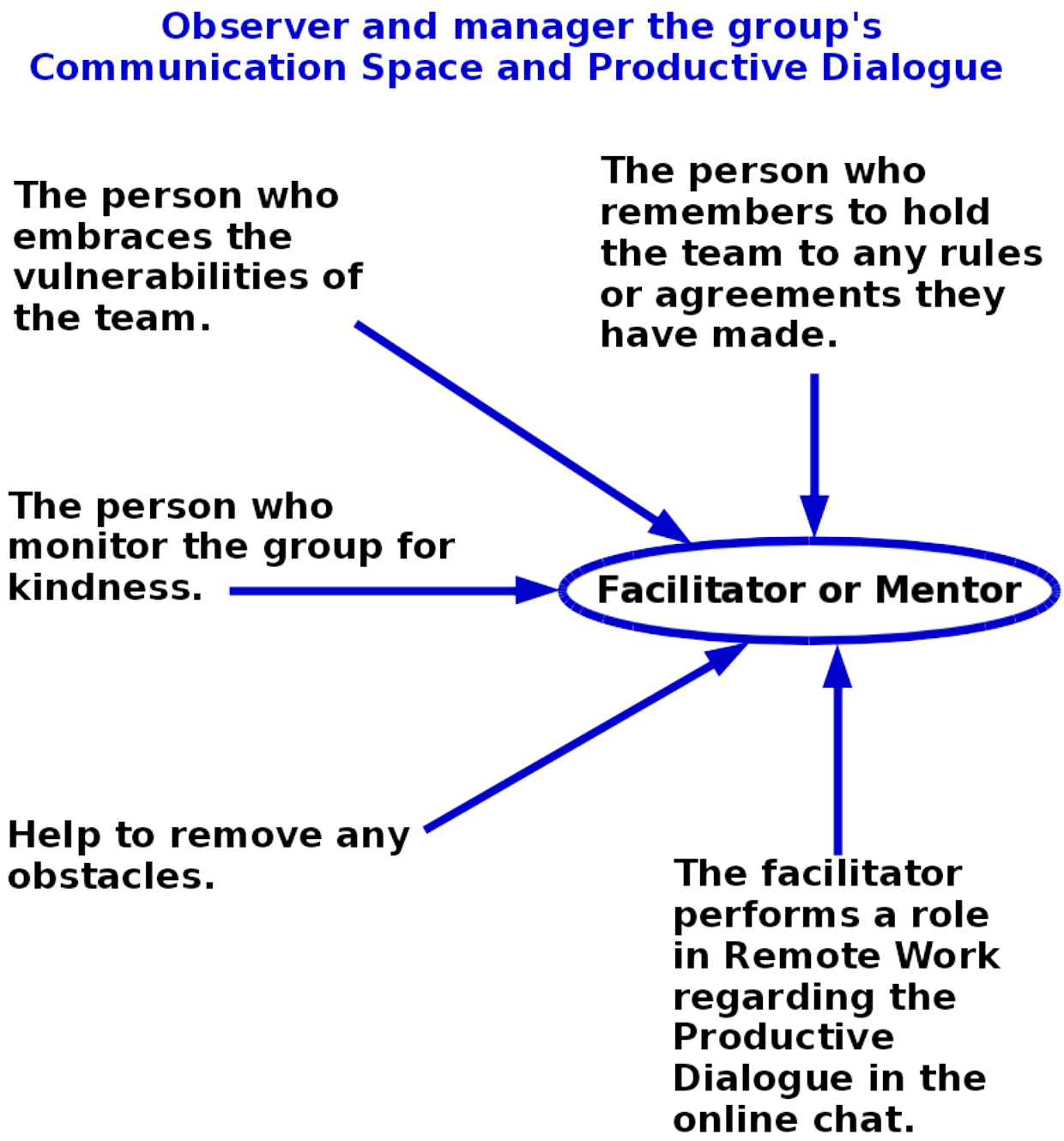


Figure 4.4: *Key points about the Facilitator role emerged through grounded theory when investigating underlying categories and concepts.*

Figure 4.4 illustrates some identified behaviors we consider as productive of the Facilitator, who is the person who remembers to hold the team to any rules or agreements they have made.

Therefore, the Facilitator responsibility is, e.g.,asking someone who is noticed a minor syntax improvement to hold that thought and wait until the tests are passing. Guarantee the Navigator being the lightning arrester for the Mob Programmers' ideas.

While the driver and navigator focus on solving the issues of the programming, the Facilitator observes and manages the group's Communication Space and Productive Dialogue. When the tests are passed, the Facilitator is responsible for increasing the team morale. The Facilitator does not need to be a stand-up comedian but only remembers to ask everyone: "Are we happy with the code as it is? Can anyone suggest a refactoring at this point?". The objective of doing this is to contribute to a safer and more innovative Communication Space.

This is also the person who remembers to hold the team to any rules or agreements they have made. For example, asking someone who is noticed a minor syntax improvement to hold that thought and wait until the tests are passing.

They can also monitor the group for kindness and respect. Sometimes the problems we face as a mob, especially when diagnosing a problem nobody understands, can cause anxiety. Under these circumstances, the facilitator role is to notice this and guide the group back to a kind and respectful dynamic.

The facilitator performs a role in Remote Work regarding the Productive Dialogue in the online chat. The facilitator helps the team set time limits for each task. Facilitator Keep the team focused on the goal and contributes to removing any obstacles and addressing any team dynamics. Also, the Facilitator protects the team from any outside interruptions and distractions.

As shown in the following quotation, the Mob Programming Team Role of the Facilitator is grounded in the data from questionnaires, interviews, and observations.

> *"On our remote mob, the facilitator can also play a great role in live-Slacking (or whatever is your insta-message tool of choice) the intent, activity and decisions of the mob. By communicating like this, other team members who join the mob later can more easily pick up the context, or even offer suggestions or advice from afar, without interrupting the flow."* P47

> "| *... The role of the facilitator was played by the agile coach, who conducted the retrospectives and controlled pilot rotation, together with keeping track of time.* |" P189

> "| *... Standard Pilot/Co-pilot and me acting as a facilitator (Director of Engineering)...* |" P64

> "| *... the facilitator helped the team set time limits for each task...* |" P111

> "| *... Facilitator Keep the team focused on the goal and contributes to removing any obstacles and addressing any team dynamics...* |" P34

> "| *... the Facilitator protects the team from any outside interruptions and distractions...* |" P61

4.5 Researcher or Scout

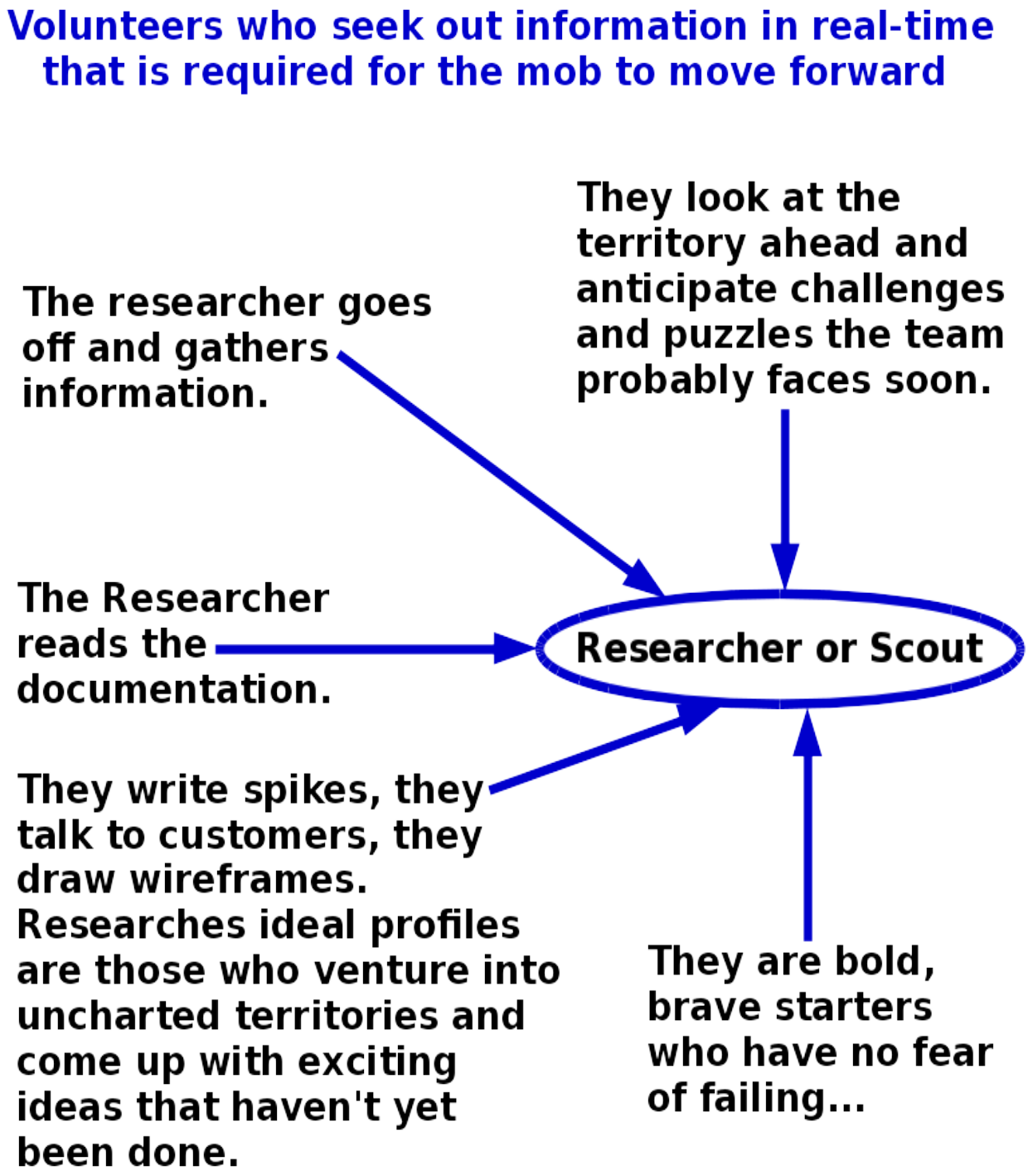


Figure 4.5: *Key points about the Researcher role emerged through grounded theory when investigating underlying categories and concepts.*

Figure 4.5 illustrates some identified behaviors we consider as productive of the Researcher, who is the volunteer who seek out information in real-time that is required for the mob to move forward.

The Researcher is curious by nature. In other words, this role fits better for someone who loves to explore. They look at the territory ahead and anticipate challenges and puzzles the team probably faces soon.

They go off and gather information: they read documentation, they write spikes, they talk to customers, they draw wireframes. Researches ideal profiles are those who venture into uncharted

territories and come up with exciting ideas that haven't yet been done. They are bold, brave starters who have no fear of failing.

The Researcher role can be done by someone on the team who has used the law of two feet to make the best use of their time. Or, they check out of the mob for a time and explicitly go and focus on that task. Or, they research in solo time to find the next best thing customers are waiting for. With innovative and fresh ideas, they make future success possible.

Researching is work people can sometimes feel guilty about, e.g., imposter syndrome, but it is vital for keeping the Mob Programming session flowing well. They are volunteers who seek out information in real-time that is required for the mob to move forward. They write spikes, they talk to customers, they draw wireframes. Researches ideal profiles are those who venture into uncharted territories and come up with exciting ideas that haven't yet been done. The Researcher reads the documentation.

Table 4.5 Mob Programming Team Roles of the Researcher or Scout, with its interactions in real-time at Mob Programming session among the team members. We are detailing the Role of the Researcher in a Mob Programming session to help other software developers start using Mob Programming or in the hope that the experienced ones contribute with this ontology to aid our software development community to have a safer and more innovative Communication Space.

Team Role	**Definition**	**Interact with, in real-time at Mob Programming session**
Researcher or Scout	The Researcher/Scout is The volunteer who seek out information in real-time that is required for the mob to move forward.	**Navigator or Co-Pilot, Team**

Table 4.5: *Mob Programming Team Roles of the Researcher or Scout, with its interactions in real-time at Mob Programming session among the team members.*

As shown in the following quotation, the Mob Programming Team Role of the Researcher is grounded in the data from questionnaires, interviews, and observations.

> *"Mobs work best when they know what to do, and are creatively, synergistically generating solutions for how to do it. When the mob gets stuck, it's often due to a lack of information. We can't decide whether to go one route or the other, because we haven't tried either of them yet and don't know enough about the implications of that choice. Sometimes we see a fork in the road and, although what is down the fork looks attractive, we do not have confidence about how long it will take us to go that route. So we stick to the routes we know. This might be a missed opportunity."* P47

> "| *... Parallel research streams were allowed...* |" P101

> "| *... We work better when know exactly what to do. Thus, the researcher was an important role...* |" P23

4.6 Quality Analyst or Quality Guardian

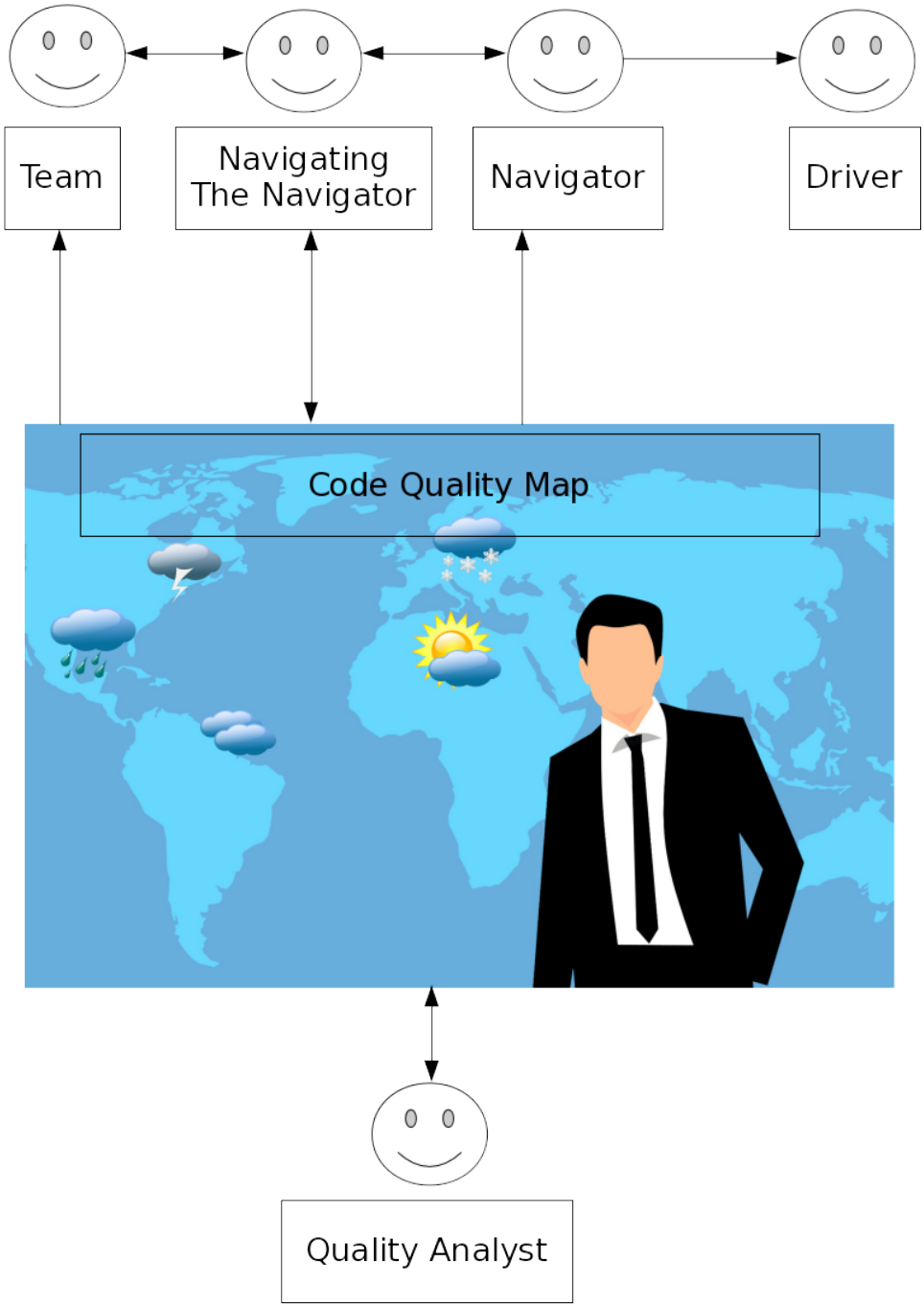


Figure 4.6: *Quality Analyst team role, responsible for the creation of a Code Quality Map. Map Photo by Jmohamedhassan from Freerange Stock.*

Figure 4.6 differs from the previous ones because it illustrates asynchronous communication beyond real-time communications. As shown in the following quotation, the Mob Programming Team Role of the Quality Analyst is grounded in the data from questionnaires, interviews, and observations.

> "| *... The Quality assurance analyst played the team role of Quality Guardian. He brought functionality validation scenarios, in addition to reinforcement of the need to build unit tests ...* |" P189

4.7 Nose or Housekeeper

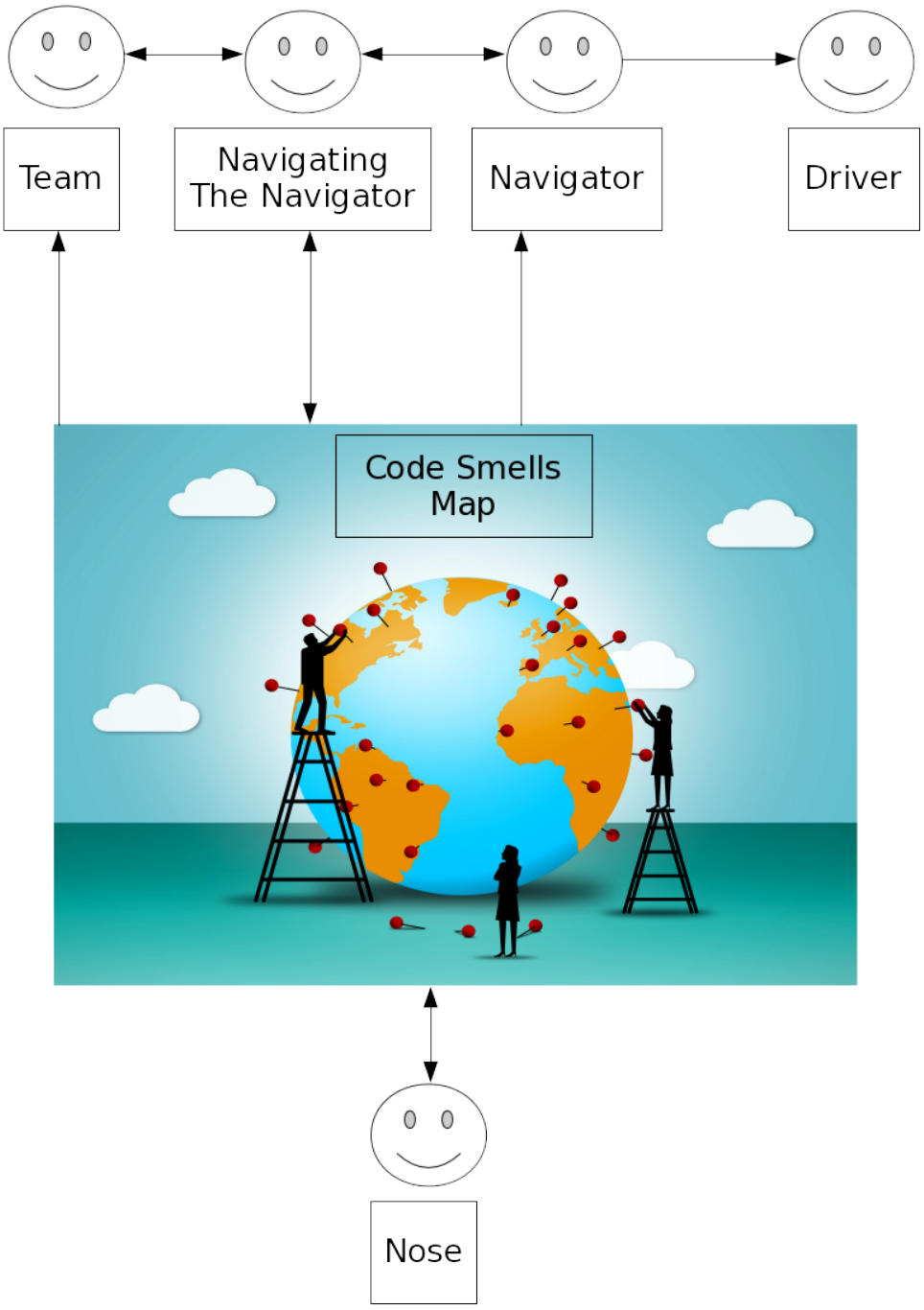


Figure 4.7: *Nose team role, responsible for the creation of the Code Smells Map. Map Photo by Jack Moreh from Freerange Stock.*

Figure 4.7 shows an asynchronous communication using a Map which permits not stopping the whole group when finding some code smell or quality problem. At an appropriate moment, they could fix it in the future using the Map.

As shown in the following quotation, the Mob Programming Team Role of the Nose is grounded in the data from questionnaires, interviews, and observations.

> *"During a mobbing session where we know something needs refactoring, but we want to stay focussed on getting to green first. We don't always get time to come back to these TODOs during that session. The Housekeeper is a bit like the Scout, in that they spend time working around the mob, keeping the flow going by tidying up these little bits of detritus flung out of the mob's productive core. While the scout looks ahead and breaks down new territory, the housekeeper focusses on making the existing code more habitable, easier to navigate and use."* P47

4.8 CyberSecurity Analyst or Hacker

Figure 4.8 shows an asynchronous communication using Security Lists and Hardening CheckLists. Human Mob Programming is working on security issues instead of an Automated CyberSecurity Risk Management tool.

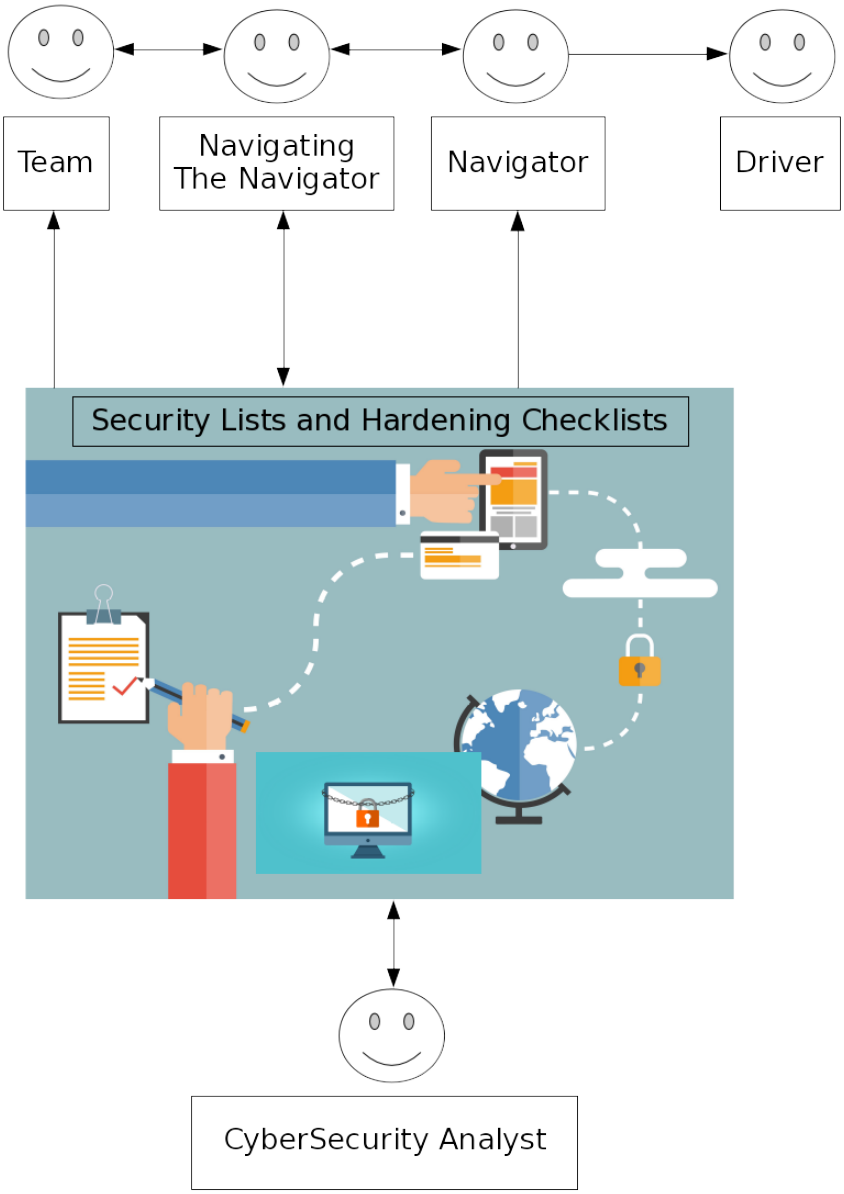


Figure 4.8: *CyberSecurity Analyst team role. It is responsible for the creation of Security Lists and Hardening CheckLists. Computer with a PadLock, Photo by Jack Moreh from Freerange Stock. Map Photo by BlueOrange from Freerange Stock*

4.9 Discussion

This section discusses our results in light of related literature to help us in the quest for explanations. Following the Grounded Theory research method. First, the data was collected and then analyzed. The intention of reviewing the literature after the results is for:

- Protection of the findings from preconceived notions;
- Improve the correlation of the research findings with the related literature by integrating both ideas.

To improve the correlation of the research findings related to Source Code Quality Maps, Security Lists, and Hardening Checklists with the related literature about Information Flow and Knowledge Sharing from da Silva and Agusti-Cullell (2008) by integrating both ideas. In agreement with da Silva and Agusti-Cullell (2008) Maps, can be considered from other perspectives. A specific map is an off-the-shelf Communication Space.

4.9.1 Quality Map

In the Future Works section of the doctoral thesis of Tonin (2018) throughout the development of her research, some gaps and opportunities for new research became evident. Software quality map to provide a mapping to connect the technical debt items with the parts of the software that contains the technical debt. Also, showing which part of the business is affected by that part of the software. This research can help teams to create a Code Quality Map and Code Smells Map toward a technical debt rank where software developers' teams can prioritize paying those debts that are affecting the most crucial part of the software.

4.9.2 Team Roles

An expressive number of researchers have explored Team Roles and their dynamics. Belbin (2012) does an overview of the development of team roles, ranging from formal job titles arranged in strict hierarchies to interdependence and cooperation between members. Various operational strategies are laid out that provide ideas and techniques that can be learned to advantage.

The findings of Sawyer *et al.* (2010) indicate that varying patterns of social interactions lead to differences in information systems development team performance. Second, their results illustrate that particular measures of information systems development performance are a simplification where a distorted impression is given. Moreover, that multiple measures of information systems development performance are unlikely to agree.

Dagenais *et al.* (2010) theorize that there are three primary factors that impact the integration experience of newcomers: early experimentation, internalizing structures and cultures, and progress validation.

Using interview and log data from 38 new product team managers and 15 team members, Ancona and Caldwell (1988) identify a set of activities that group members use to manage their dependence on external groups. Team members carry out scout, ambassador, sentry, and guard activities along with immigrant, captive, and emigrant roles to manage external transactions. It is hypothesized that high team performance is associated with a fit between the level of boundary activity and the degree of resource dependence.

Ancona and Caldwell (1988) and Sawyer *et al.* (2010) describe five boundary-spanning roles have been identified as means to encourage communication across boundaries: ambassador, scout, guard, sentry, and co-ordinator. An ambassador represents the team to external stakeholders and persuades them to support the team. A scoutis responsible for scanning within and outside their organizations for new ideas and technologies. In Mob Programming teams, on the other hand, learning new technologies and concepts is a continuous effort performed by all team members.

Team roles have been described by Ancona and Caldwell (1988), Belbin (2012), and Sawyer *et al.* (2010) outside the field of software development . Belbin (2012) suggested roles based on individual

behaviours traits of team members. In contrast, the Mob Programming team roles are focused on self-organization.

Ancona and Caldwell (1988) and Sawyer *et al.* (2010) describe five boundary-spanning roles focused on encouraging communication of the team with external stake-holders. An ambassador role in their study represents the team to external stake-holders and persuades them to support the team.

4.9.3 Team Strategies toward a good performance

The Mob Programming team roles identified in this research include internal, external, and interfacing roles. The major part of the Team Roles are primarily internal facing, the External Members are external facing, and the Dev Team are interfacing roles between the team and their external stake-holders (senior management and customers). The self-organizing nature of these roles identified in this research is further consolidated when compared to the characteristics of self-organizing teams defined by Anderson and McMillan Anderson and McMillan (2003).

Anderson and McMillan (2003) examine fundamental issues of teamwork, and question whether such teams really are analogous. After detailing some of the striking similarities, they conclude that they are indeed equivalent and comparable. Thus, their work is a preliminary study into whether nature-specifically, insect societies-may provide not just a valid metaphor but, moreover, a model for organizational shift and effective function in human enterprise.Thus, a possible metaphor about the strategy of only sometimes work with the entire team, are the bees that do not swarming all the time, but only in specifics moments: lack of space for posture or to store the honey and pollen, etc. Other possible metaphor is collective sports, one example of sport with ball is the soccer, although the focus of the game is around the ball, in a good team always has players playing without the ball, following tactics functions. Therefore, these tactics functions are also fundamental for Mob Programming teams.

Western Eletric made experiments attempted to determine the relationship between light levels and worker efficiency. The data compiled by the Illumination Experiments indicated only a minor correlation between light levels and worker productivity. However, one interesting observation is sometimes when increase the light level, the productivity grow up and when decrease the light level the procuctivity grow up too Mayo (1933).

A possible explanation is an external factor that in some way improve the bond formed among the members maybe could be positive to the productivity of the team. But, metaphorsly by the last time, medicine is poison depending on the dose.

4.10 Well-Defined Team Roles Theory

This section presents and describes the gathered results of the GT study conducted about Mob Programming Team Roles, in the form of our Six C's model of Well-Defined Team Roles Theory elaborated in terms of the Six C's theoretical model for the category Well-Defined Team Roles.

The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;

2. **Conditions**: Factors that are prerequisites for having Well-Defined Team Roles are: Planning or Talking before the Mob Programming session about what roles each one prefers to perform;

3. **Causes**: The reason is the Programmers decide to adopt the Mob Programming Team Roles Ontology, choosing the appropriate Roles for their context;

4. **Consequences**: Outcomes or effects of having Well-Defined Team Roles, that is, a contribution to all team members keeping the focus on problem-solving;

5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. Driver Rotation, Team Roles Rotation, Members Rotation among the teams, and Retrospectives;

6. **Covariances**: Correlations between different categories: Well-Defined Team Roles vary with Group Size and Priorities. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

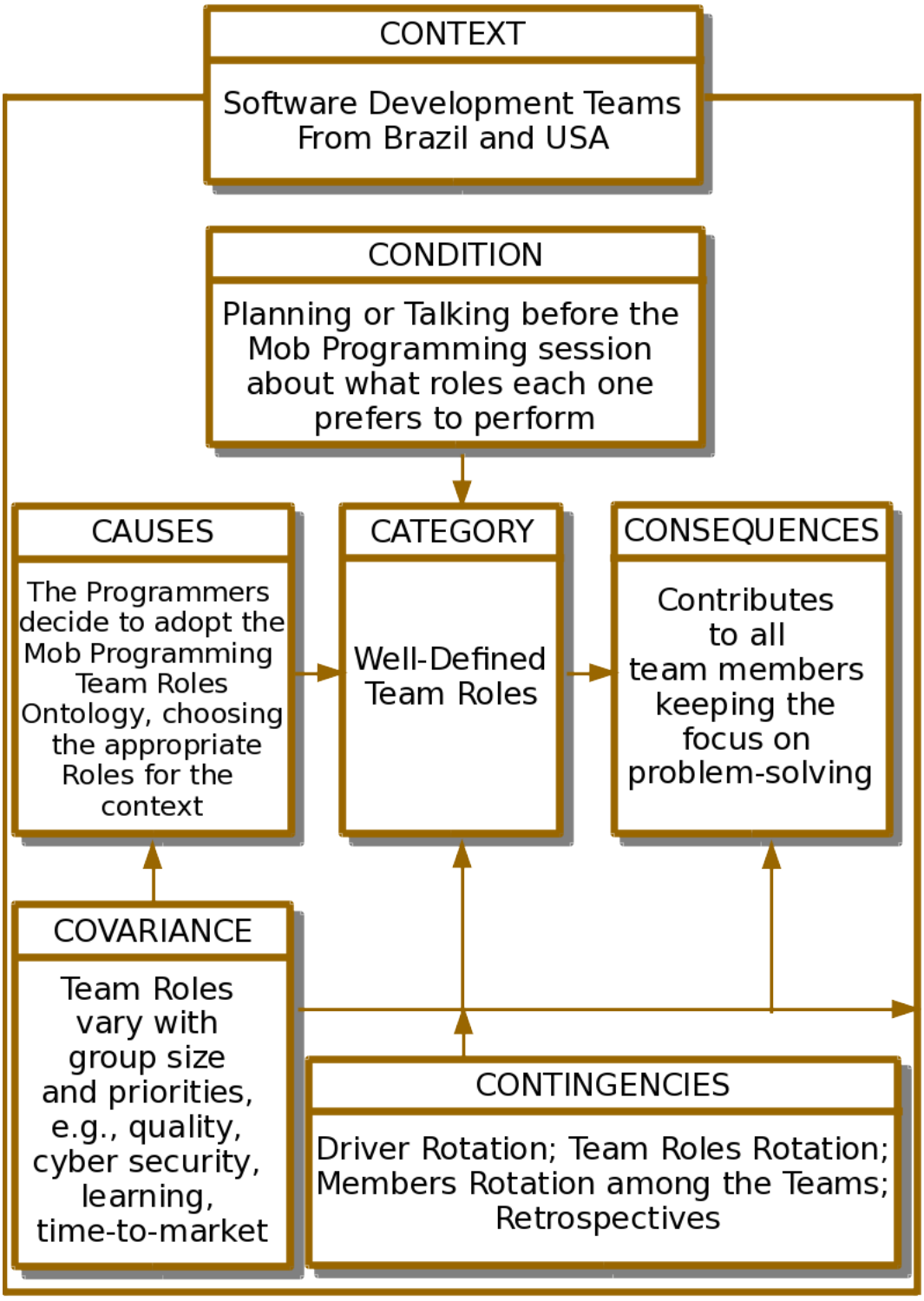


Figure 4.9: *Six C's model of Well-Defined Team Roles Theory.*

The Six C's theoretical coding paradigm was used for presenting the results in Figure 4.9. It is

SIX C's model of our Six C's model of Well-Defined Team Roles Theory. Figure 4.9 to illustrate our Well-Defined Team Roles Theory. The category Well-Defined Team Roles is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

4.10.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

4.10.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., there is planning or at least a talking before the Mob Programming session about what team roles each Mob Programmer prefers to perform. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

4.10.3 Causes

Causes define the reasons for the core category to occur. The cause to have a Well-Defined Team Roles is the Programmers decide to adopt the Mob Programming Team Roles Ontology, choosing the appropriate Roles for their Context.

4.10.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of having a Well-Defined Team Roles is that contributes to all team members keeping the focus on problem-solving.

4.10.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. Four different contingencies were identified:

- Driver Rotation;
- Team Roles Rotation;
- Members Rotation among the Teams;
- Retrospectives.

4.10.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Well-Defined Team Roles vary with Group Size and Priorities, due the following reasons:

- If the teams prioritize quality. And the Group Size permits two members to focus on quality. Thus, one member may perform the Role of Quality Analyst, and another be the Nose;
- Else if the team prioritizes the CyberSecurity and the Group Size permits to have one or two members focus on quality and one other exclusive for CyberSecurity. The team could have both because the group size permits;
- Else case, the team prioritizes both, Quality and CyberSecurity. And it has only three members. Thus, the Navigator could also perform the CyberSecurity, and The Quality Analyst performs the Role of Nose too;
- Otherwise, there are many possibilities depending on the Group Size and priority of the team. Moreover, having an ontology helps everyone learn and decides what role prefers to perform. Additionally, of course, guidance the team chooses with greater accuracy the ideal Group Size for their respective Team Role needs.

4.11 Conclusion

When the team uses mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Was observed a correlation with the challenge of keeping the whole Mob Programmers focused on solving the problem.

> "| *We principally use the roles of Pilot and Navigator. The idea is that all members are always aware of the code quality. Thus, it is expected the produced codes are within the established standards. However, if something is done outside the code standard, the Navigators seek to correct it immediately. If there is doubt, some navigators seek to do research in parallel with what is being at the moment developed. I believe that, although the pilot has the keyboard, this limits him a little in terms of the logical development of the system since he has to worry about writing the code itself. Navigators, on the other hand, have a wide range of possibilities for action, having more freedom of discussion, research, and exchange of ideas.*|" P66

Following is the team answer for the question "What were the difficulties when starting to use Mob Programming?". This is the same team that use mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined.

> "| *The main difficulty is keeping the entire group focused on solving the problem.*|" P66

The other Mob Programming teams used more team roles than only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Additionally, these other teams reported distinct difficulties in beginning with Mob Programming when answering the question, "What were the difficulties when starting to use Mob Programming?". Moreover, none of these other groups claims about the problem of the difficulty of keeping the entire group focused on solving the problem.

Toward being disciplined with the thesis structure. We detailed in the chapter "Literature Review" the literature on Pair Programming, and in the session 2.2.5 "Mechanisms to improve the performance of pair programming" that argues some behaviors to Navigator toward the Pair Programming session be productive and not lose the focus. What corroborates with our thesis about well-defined roles for all team members helps everyone maintain the focus.

To summarize, Well-Defined Team Roles are the Programmers when the team decides to adopt the Mob Programming Team Roles Ontology, choosing the appropriate Roles for their Context. It observed the positive effect of having Well-Defined Team Roles is helps all team members keep the focus on problem-solving. As shown above.

Chapter 5

An Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors

This chapter presents the core of our grounded theory of An Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors. The theory explains how software development teams perform balancing acts on a set of integrated practices, in order to become an effective Mob Programming team. The integrated practices are Driver Rotation, Retrospectives, Structure breaks, Group Size, Room setup, and Automation. This chapter also describes the informal behaviours which exist in Mob Programming teams.

By explicitly acknowledging this Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors, the whole team could suggest people's behaviors to better contribute to the overall effectiveness and flow of the Mob Programming Session and the Communication Space.

Figure 5.1 shows the emergence of a grounded theory of Mob Programming team roles from underlying categories and concepts. Some of the Mob Programming roles are transient because, unlike organizational roles, they emerge in response to challenges faced by the Mob Programming session and disappear or become dormant as the problems subside. The Mob Programming team roles are spontaneous because unlike organizational roles. They are intuitively picked up by different members of the team.

Figure 5.1 also provides an overview of Mob Programming team roles that we observed as more stable. The following sections describe each of these Mob Programming roles in detail. The descriptions include selected quotations drawn from the interviews that shed particular light on these categories and are spread across participants and by their organizational roles.

The quotations are presented verbatim from the interview transcripts or the questionnaire answers. When they are with square brackets symbol "[ ]" is used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Ellipsis, a punctuation mark "..." used within quotations to indicate that something has been left out, i.e., to indicate an omission from speech or writing of a word or words that are unnecessary to the specific point or able to be understood from of the context presented, the forfeit of one or more items from a construction to avoid repeating the identical or equivalent items that are in a preceding or following making. While double three full stops preceded and followed by spaces " ... ... " indicate combining two sentences referring to the same context but derived from different parts of the same interview or questionnaire. The quotation between pipes, as known as vertical lines, or vertical slash, "| |" means a translation from Portuguese occurred in the sentence.

5.1 Introduction

We begin by presenting the notions of Communication Space and Productive Dialogue. The reason is they will be used in the subsequent sections and chapters. Then we describe the practices that contribute to a productive Mob Programming session. Afterward, we detail the behaviors of Mob Programmers observed as fruitful. We connected in this chapter what has been observed in practice with related literature that helps us with the explanation. Afterward, a discussion on the theory of planar graphs can be related in some way to the positioning of Mob Programmers. At the end of this chapter, we present the 6 C's model.

5.2 Communication Space

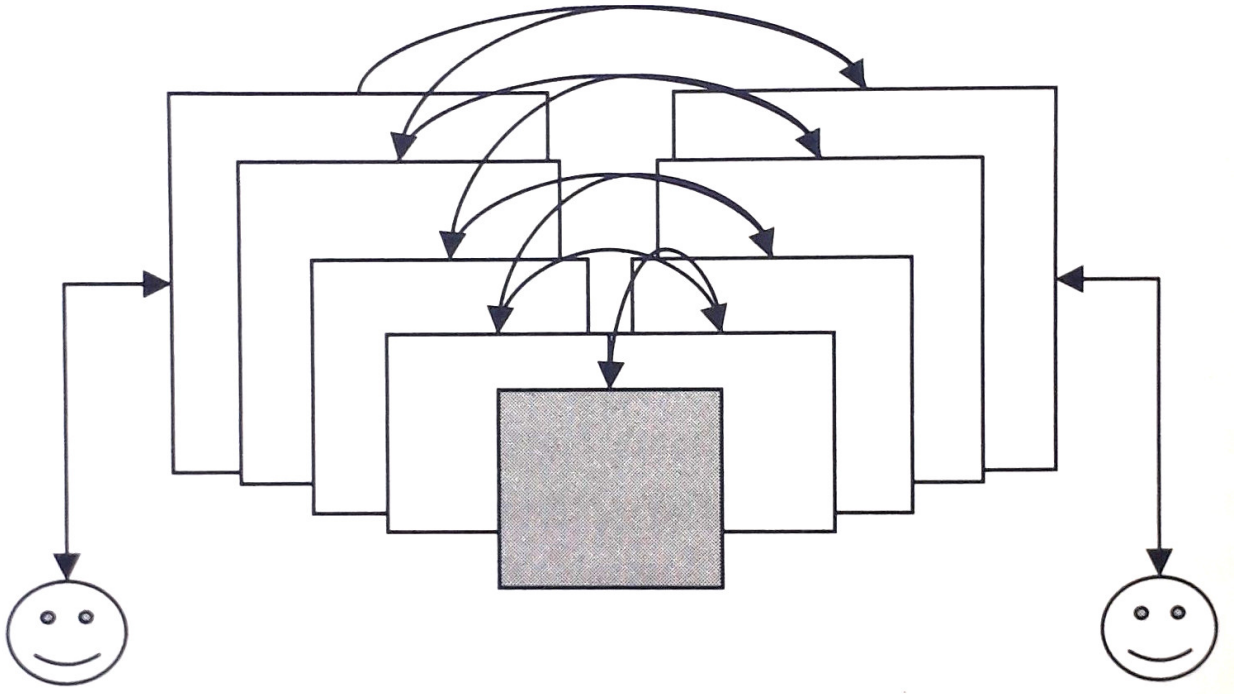


Figure 5.1: *Collaborative construction of a communication space from da Silva and Agusti-Cullell (2008).*

da Silva and Agusti-Cullell (2008) a model to simplify the description of their notion of *shared understanding*, that is, **Communication Space** built collaboractively by the agents engaged in a specific dialogue. Note that their theory is from Artificial Intelligence field. Moreover, the first author is my advisor, and I am adapting this ideas to help me understanding and explain the Mob Programming approach. da Silva and Agusti-Cullell (2008) defines Communication Space as:

> **"A Communication Space is an agreed-upon configuration of terms to be shared by a group of agents, through which they interact in such a way that the consequences of the interaction are (at least partially) predictable for all members of the group (Figure 5.1)".**

da Silva and Agusti-Cullell (2008) gives a concrete example that inspired me, the interaction between students and a lecturer during a course at University. During the first moments of the beginning, the lecturer and the students do not know each other. Thus, the lecturer can make some assumptions about the students, considering the point of the course they are; the students can also make some assumptions about the lecturer, and both lecturer and students can make assumptions about the topics to be covered during the course. It is perfectly possible, however, that these assumptions do not match at all with each other or with reality.

During the course, the students and the lecturer interact during their lectures through assignments, lecture notes, and so on. If everything goes all right, by the end of the course. Therefore, the lecturer and the students will have built partial views about each other and related to the topics covered during the classes, which will at least trigger appropriate behavior during an interaction, e.g., when the lecturer presents a problem to a student, the student shall make use of the suitable topics covered during the lectures to solve it and deliver the correct answer to the lecturer. Notice that the student and the lecturer don't need to have the same understanding of the topic used to solve the problem or of the problem being solved. As can be frequently observed in practice, it is also not necessary that students have the same opinions about a lecturer as the lecturer has regarding himself, nor that the lecturer has the same judgment about the students as they have about themselves. It is only required that these opinions **trigger compatible behavior** during the **interactions** between the lecturer and the students.

When the course finishes, the lecturer and the students must have collectively built a **Communication Space**. Also, the **Communication Space** here in this example is characterized by the part of the technical information delivered by the lecturer that has been effectively fully understood by the students. According to da Silva and Agusti-Cullell (2008), when all agents in a group access the same **Communication Space**, they can engage in a **Productive Dialogue**.

5.2.1 Productive dialogue

A Productive Dialogue is such that the participation agents can coordinate their actions. The agents participating in a productive dialogue must not necessarily act together to reach a common goal. Indeed, they must not share any objective at all. What characterizes a fruitful dialog is the truth that the participating agents in the dialogue have, due to the interaction with the other agents, more and better means at hand to reach their own goals da Silva and Agusti-Cullell (2008).

There are two perspectives upon which **productive dialogues** can be analyzed:

- The individualistic perspective, based on the perceptions of the dialogue realized by each individual agent participating in the dialogue;
- The social perspective, based on the global result of the interactions among the agents participating in the dialogue.

Through the individualistic perspective, da Silva and Agusti-Cullell (2008) analyzes what can be achieved by an individual agent which would not be achievable for that agent without the participation of other agents. Through the social perspective, we consider the agents participating in the dialogue as a **group** and analyze the result of their collective actions and perceptions.

What characterizes a productive dialogue is the fact that the participating agents intentionally acknowledge the participation of each other in the dialog. This mutual acknowledgment is identified by the shared communication space built by these agents. According to da Silva and Agusti-Cullell (2008), affordance is an attribute to limit the uncertainty involved that is present in any interaction, including productive dialogs.

Productive dialogs are, therefore, the distinctive mark of social networks built for specific purposes and to reach specific goals. It is through fruitful dialogues that people interact in everyday life da Silva and Agusti-Cullell (2008).

The social and the individualistic approaches to analyzing dialogues and productive dialogues complement each other. When designing systems to enable dialogues or analyzing posteriorly emergent systems and their consequences, these two approaches should preferably be considered da Silva and Agusti-Cullell (2008).

5.3 Driver rotation techniques, as Facilitators for productive dialogues

According to da Silva and Agusti-Cullell (2008), a Facilitator for productive dialogues is a conceptual tool made of three components:

- A set of stereotyped roles for the participants in the dialogue;
- A set of initial information systems for each of the participants in the dialogue, effectively adopted by the agents to the construction of the Communication Space through which the productive dialogue shall occur;
- Standardzed procedures for interaction in order to incrementally build the Communication Space for the productive dialogue.

According to da Silva and Agusti-Cullell (2008), facilitators for productive dialogues can be implemented in several forms, depending on their intended use and users. In agreement with my advisor following are describe three alternatives to do the driver rotation in a Mob Programming section. The first is a stereotyped role to help the participants (Mob Programmers) in the dialogue (Mob Programming session). The second is a simple software (information system) effectively adopted by the agents to bootstrap the construction of the Communication Space through which the productive dialogue shall occur. The third is a standardized procedure for helping the interaction and incrementally building the Communication Space for productive dialogue.

5.3.1 Facilitator

> "|*We used a pre-defined rotation time controlled by the facilitator. During the retrospective sessions, we had an analysis in the time [for Drive Rotation], initially 10 minutes for pilotages, and it was increased to 15 minutes after the first retrospective.*|" P31

5.3.2 Initial Information System

Timer

> "*We used the clock, but we were not very strict about it. Sometimes we lost track of timing so we would decide for rotation.*" P1

> "*We created a simple script to count down the time of each driver and alert the team when it's time to swap.*" P12

> "*We developed our own timer tool. It received an integer as argument and warned us when time was up. We enjoyed the timer because it helped us organized the rotation. We typically used 15 20 minutes.*" P16

> "*We programmed an adapted timer that we downloaded from GitHub. [However,] after a retrospective, we decided to rotate the keyboard more organically, without a fixed amount of time to be at the keyboard.*" P103

5.3.3 Standardzed Procedures

Strong Style

> "*We use the Strong Style. In the Strong Style keyboard rotation, if the pilot wants to give an idea, he needs to hand the keyboard to someone else. In other words, the pilot has no arguments, and for an idea to pass from a head to the computer, it must pass through someone else's hands.*" P5

> "*We do not use 100% of the Strong Style. [However,]the initial idea was that it would be like this, but when someone more experienced was in the pilotage, he ended up contributing with ideas and suggestions.*" P111

> "*We were able to use the Strong Style at some points. More, in the beginning, because we soon decided to try our style.*" P52

Red/green Criteria

> "*Instead of a fixed timebox, we would [rotate the driver] with a red/green criteria refactor. [Moreover,] only switch after a few test cases were passing and at least one new failing test was created.*" P184

5.4 Team Members Rotation among the distinct Mob Programming sessions

> "*Team Members Rotation among the distinct Mob Programming sessions. If a team member is not contributing, it should be moved to another Mob Programming session.*" P194

5.5 How many programmers in a session of Mob Programming

> "*It is variable size, depends on the complexity involved.*" P188

5.6 Remove one person and team performance

> "|*Due to the characteristic of being in the military, it is usual that, occasionally, one or more members have to be absent. In general, it does not cause an impact, but it can be observed that when more experienced members are away, e.g., on a mission, tasks could take longer to be resolved.*|" P50

5.7 Remove more than one person and team performance

> "|*At some point, more than one person needed to be absent, and the principal impact is that tasks can sometimes take a little longer to be performed.*|" P50

5.8 Retrospectives

Retrospectives helps to improve the Mob Programming session.

> "|*One of the changes resulting from a mini-retrospective was the revision of the piloting time, which were very useful for adjusting the route. We used a pre-defined rotation time controlled by the facilitator. During the retrospective sessions, we had an analysis in the time [for Drive Rotation], initially 10 minutes for pilotages, and it was increased to 15 minutes after the first retrospective.*|" P31

> "|*Yes, we adapt the mob programming to our needs, adjusting without limiting ourselves to the initially established rules.*|" P83

5.9 Structure breaks

> "*I believe that creating "slack" for people to think and the process sometimes brings up that breakthrough idea that solves or simplifies the whole solution. I haven't quantified/documented the [exact number] of times this happened with our teams after a break, but this happened indeed, whether it is this you want to know and it is trying to discover.*" P137

5.10 Programmer Satisfaction

Behavioral positive response to the satisfaction of programmers Contributing to managing the complexity and remote work.

> "|*I was satisfied with Mob Programming, however, I understand that there are times when it is better/more productive to work alone or in pairs. The positive points are the sharing of knowledge, team integration, and learning. As negative points, I mention the team's fatigue after the end and the reduced speed, especially due to the pilot changes. The observation of how each pilot codifies and the sharing of individual knowledge transforms it into collective knowledge. Yes, all members approved and were satisfied with the mob programming.*|" P144

> "|*For complex tasks, the satisfaction was unanimous. For simple tasks, it is more efficient to do them individually or in pairs. The main plus point is the quick transfer of knowledge. The main negative point is the long delay in the problem-solving process. Being able to discuss possible solutions with members during real-time development. The approval was unanimous for the dissemination of knowledge and discussion of complex problems.*|" P150

> "|*I am satisfied with Mob Programming and Remote Mob Programming.* |" P22

> "|*At some point, more than one person needed to be absent, and the principal impact is that tasks can sometimes take a little longer to be performed.*|" P50

5.11 Kind and Fearlessness Programmer

The courage of programmers is a behavioral positive response to Contributing to managing the complexity and remote work. Kindesse help to create a safe Communication Space, allowing the team members showing theirs vulnerabilities.

This is crucial because always the programmers need to learn something new. And, expose what the person do not knows requires vulnerability. When a programmer do not know the programming language being used at a moment when this person is the Driver, a fearlessness attitude helps because all team members contributing this programmer learn quicker. Contributing to creation of a Safer, Innovative Communication Space wherer people Embracing the Vulnerabilities with Kindness.

5.12 Room Setup

A **graph** is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects of the graph correspond to vertices (nodes) and the relations between them correspond to edges. A graph is depicted diagrammatically as a set of dots depicting vertices connected by lines or curves depicting edges. Formally,

> "*A graph* $G = (V, E)$ *consists of* V *, a non-empty set of vertices (or nodes) and* E *, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints.*"

Figure 5.2 illustrates a graph K6 complete with 15 edges. Suppose their nodes are the Team Members and the edges are a Pattern of Interaction. Thus, there are 15 possibilities for Patterns of Interaction.

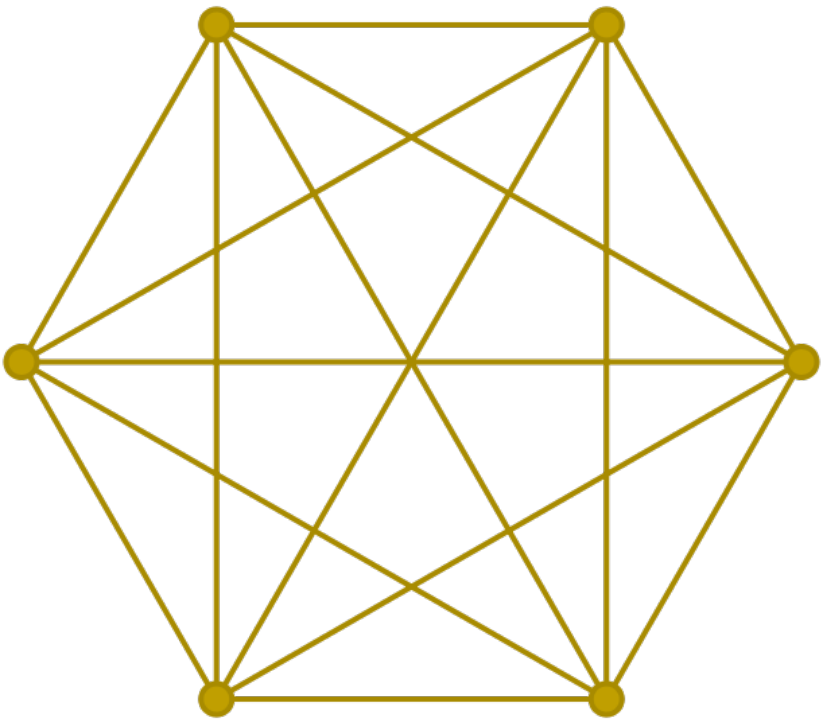


Figure 5.2: *K6 complete*

Figure 5.3 illustrates a graph K7 complete with 21 edges. Suppose their nodes are the Team Members and the edges are a Pattern of Interaction. Thus, there are 21 possibilities for Patterns of Interaction.

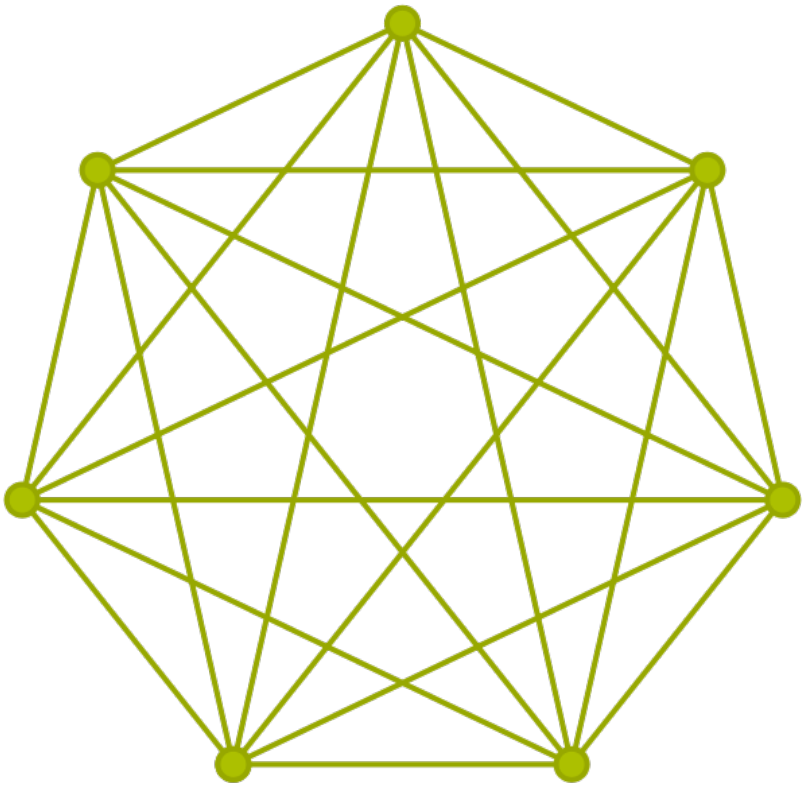


Figure 5.3: *K7 complete*

Room Setup must consider the Group Size. Figure 5.2 and Figure 5.3 illustrate the exponential increases in the number of possibilities of Patterns of Interaction.

5.12.1 Real-Time Communications

The vocal folds produce sound when they come together and then vibrate as air passes through them during exhaling air from the lungs. This vibration creates a sound wave for the human voice. The vocal folds must vibrate together symmetrically and regularly for the sound to be clear and not raspy or hoarse Kayes (2015).

The science of sound is acoustics which deals with the propagation of mechanical waves in various mediums such as solid, liquid, and gases. Sound is a wave defined scientifically similarly to the other mechanical waves in physics Kadam and Nayak (2016).

Figure 5.4 represents the Sound Waves. They travel through any material that can compress: air, solid, or liquid. Just remembering, the first telephones had a cross lines problem. Nowadays, an open space where many people talk together makes it harder to understand what another person is talking about.

Figure 5.5 alludes to a piece of good music where all member of a band plays or singles in perfect rhythm. Similarly, the Communication Space must make viable for the team members contributes together to problem-solving.

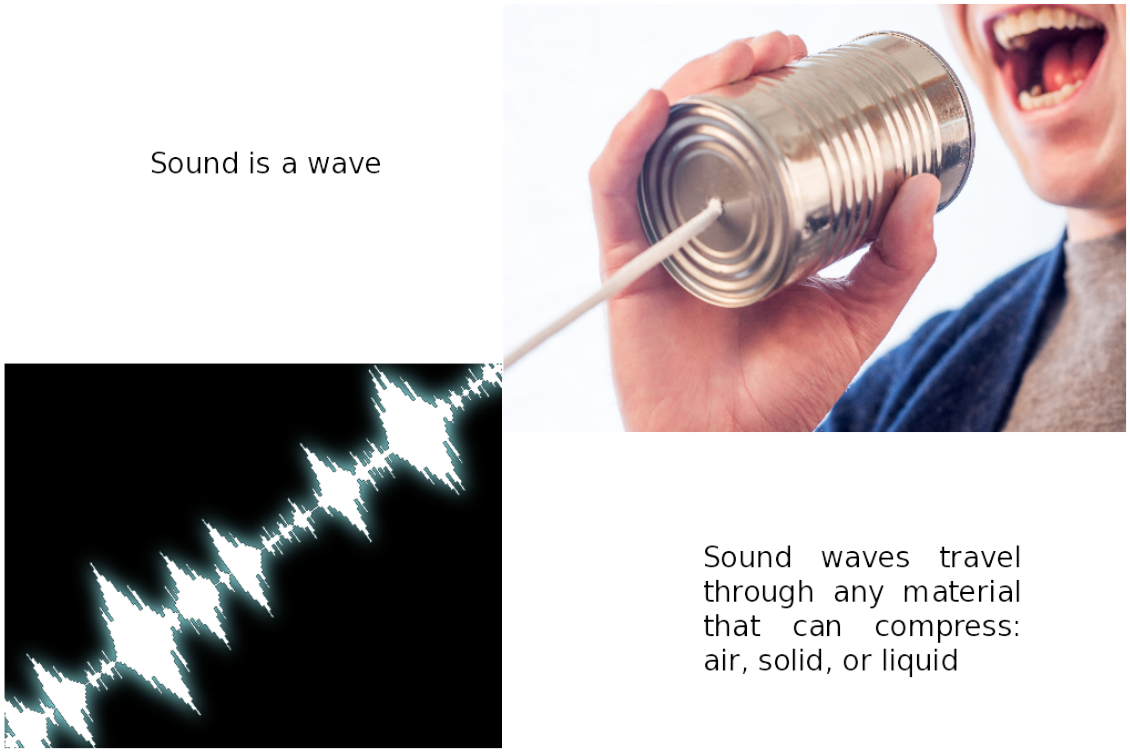


Figure 5.4: *Voice. Sound Waves. Photo of the Left by StuartMiles from Freerange Stock. Photo of the Right by Gratisography / Ryan McGuire from Freerange Stock*



Figure 5.5: *Voice. Sound Waves. Photo by StuartMiles from Freerange Stock*

5.13 Software setup

"*Microsoft Teams, Git, Visual Studio Code, Miro.*" P22

"|*Visual Studio Code with VS Code Live Share plugin.*|" P123

"|*Ruby. Ruby on Rails. Slack. Discord. Tuple. Google hangouts.*|" P99

5.14 Automation

I suggest in the Future Works session the Automation of the Driver Role through an Artificial Intelligence software listening the Navigator voice and writing the code following the patterns of codification and standard code established.

"*The first thing that comes to my mind is build time. We have been actively working on improving the build time to decrease the burden of waiting for the compilation, especially when we are mob programming.*" P162

5.15 Practices related to the COVID-19 Pandemic

Social distancing, staggered work shifts, downsizing operations, delivering services remotely, and other exposure-reducing measures were needed. Moreover, organizations suffered from increased rates of worker absenteeism. Workers were absent because they were sick; were caregivers for ill family members; were caregivers for children if schools or daycare centers were closed; had at-risk people at home, such as immunocompromised family members; or were afraid to come to work because of fear of possible exposure, as suggested by Safety *et al.* (2020).

According to Occupational Safety *et al.* (2020) from the United States Department of Labor, for most employers, protecting workers will depend on emphasizing basic infection prevention measures. As appropriate, all organizations participants of this study implement good hygiene and infection control practices, including:

- Promote frequent and thorough hand washing, including by providing workers, customers, and worksite visitors with a place to wash their hands. If soap and running water are not immediately available, provide alcohol-based hand rubs containing at least 60% alcohol.
- Encourage workers to stay home if they are sick.
- Encourage respiratory etiquette, including covering coughs and sneezes.
- Employers should explore whether they can establish policies and practices, such as flexible worksites (e.g., telecommuting) and flexible work hours (e.g., staggered shifts), to increase the physical distance among employees and between employees and others if state and local health authorities recommend the use of social distancing strategies.
- Discourage workers from using other workers' phones, desks, offices, or other work tools and equipment, when possible.
- Maintain regular housekeeping practices, including routine cleaning and disinfecting of surfaces, equipment, and other elements of the work environment. When choosing cleaning chemicals, employers should consult information on Environmental Protection Agency (EPA)-approved disinfectant labels with claims against emerging viral pathogens. Products with

EPA-approved emerging viral pathogens claims are expected to be effective against SARS-CoV-2 based on data for harder to kill viruses. Follow the manufacturer's instructions for use of all cleaning and disinfection products (e.g., concentration, application method and contact time, PPE).

5.15.1 Implement Workplace Controls

Occupational safety and health professionals, using according to Safety *et al.* (2020) a framework called the textithierarchy of controls to select ways of controlling workplace hazards. In other words, the best way to control a hazard is to remove it from the workplace rather than relying on workers to reduce their exposure. *hierarchy of controls* are classified in *Engineering Controls* and *Administrative Controls*, they are following described.

Engineering Controls

All organizations participants adopted remote work during the COVID-19 pandemic to protect their collaborators. After the pandemic, when software developers returned to workplaces, some Engineering Controls were adopted. According to Safety *et al.* (2020), Engineering Controls isolate employees from work-related hazards. In workplaces where these are appropriate, these mechanisms reduce exposure to hazards without relying on worker behavior and can be the most cost-effective solution to implement. Some Engineering Controls for SARS-CoV-2 adopted were:

- Installing high-efficiency air filters.
- Increasing ventilation rates in the work environment.
- Installing physical barriers, such as clear plastic sneeze guards.

Administrative Controls

Administrative controls require action by the worker or employer. In Safety *et al.* (2020), administrative controls are changes in work policy or procedures to reduce or minimize exposure to a hazard. It was observed that all organizations participants implement a set of administrative controls for SARS-CoV-2 among them:

- Encouraging sick workers to stay at home.
- Minimizing contact among workers, clients, and customers by replacing face-to-face meetings with virtual communications and implementing telework if feasible.
- Establishing alternating days or extra shifts that reduce the total number of employees in a facility at a given time, allowing them to maintain distance from one another while maintaining a full onsite work week.
- Discontinuing nonessential travel to locations with ongoing COVID-19 outbreaks.
- Developing emergency communications plans, including a forum for answering workers' concerns and internet-based communications, if feasible.
- Providing workers with up-to-date education and training on COVID-19 risk factors and protective behaviors (e.g., cough etiquette and care of PPE).
- Training workers who need to use protecting clothing and equipment how to put it on, use/wear it, and take it off correctly, including in the context of their current and potential duties. Training material should be easy to understand and available in the appropriate language and literacy level for all workers.

5.15.2 Safe Work Practices

Safe work practices are a variety of administrative controls that include procedures for safe and proper work used to reduce the duration, frequency, or intensity of exposure to a hazard. In agreement with Safety *et al.* (2020) all organizations participants adopted safe work practices for SARS-CoV-2 including:

- Providing resources and a work environment that promotes personal hygiene. For example, provide tissues, no-touch trash cans, hand soap, alcohol-based hand rubs containing at least 60 percent alcohol, disinfectants, and disposable towels for workers to clean their work surfaces.
- Requiring regular hand washing or using of alcohol-based hand rubs. Workers should always wash hands when they are visibly soiled and after removing any PPE.
- Post handwashing signs in restrooms.

Following Safety *et al.* (2020), all the participants' organizations developed, implemented, and communicated about workplace flexibilities and protections:

- Actively encourage sick employees to stay home.
- Ensure that sick leave policies are flexible and consistent with public health guidance and that employees are aware of these policies.
- Talk with companies that provide your business with contract or temporary employees about the importance of sick employees staying home and encourage them to develop non-punitive leave policies.
- Do not require a healthcare provider's note for employees who are sick with acute respiratory illness to validate their illness or to return to work, as healthcare provider offices and medical facilities may be extremely busy and not able to provide such documentation in a timely way.
- Maintain flexible policies that permit employees to stay home to care for a sick family member. Employers should be aware that more employees may need to stay at home to care for sick children or other sick family members than is usual.
- Recognize that workers with ill family members may need to stay home to care for them.
- Be aware of workers' concerns about pay, leave, safety, health, and other issues that may arise during infectious disease outbreaks. Provide adequate, usable, and appropriate training, education, and informational material about business-essential job functions and worker health and safety, including proper hygiene practices and the use of any workplace controls (including PPE). Informed workers who feel safe at work are less likely to be unnecessarily absent.

5.15.3 Personal Protective Equipment (PPE)

While engineering and administrative controls are considered by Safety *et al.* (2020) as more effective in minimizing exposure to SARS-CoV-2, PPE may also be needed to prevent certain exposures. While correctly using PPE can help prevent some exposures, it should not take the place of other prevention strategies. Examples of PPE include: gloves, goggles, face shields, face masks, and respiratory protection, when appropriate. During an outbreak of an infectious disease, such as COVID-19, recommendations for PPE specific to occupations or job tasks may change depending on geographic location, updated risk assessments for workers, and information on PPE effectiveness in preventing the spread of COVID-19. All types of PPE must be:

- Selected based upon the hazard to the worker.
- Properly fitted and periodically refitted, as applicable.

- Regularly inspected, maintained, and replaced, as necessary.
- Properly removed, cleaned, and stored or disposed of, as applicable, to avoid contamination of self, others, or the environment.

5.15.4 Using Mob Programming in Post-Pandemic

Beyond the formal actions adopted by the organizations. Following are some participants saying they are comfortable with face-to-face work.

> "|*I believe that the actions adopted are sufficient.*|" P39

> "|*Initially, we had a TV and a table with a desktop, with several chairs, in a space in the room where everyone worked. Today in post-pandemic, that arrangement has continued, but in a larger room, which we call the workshop. We are already using face-to-face mob programming again.*|" P66

The participants reported feeling safe with face-to-face work, including the ones that prefer remote work.

> "|*I feel comfortable with the procedures adopted by the company.*|" P144

> "|*The company adopted remote work, face masks indoors in the post-pandemic period, and proof of vaccination for face-to-face work, among other measures.*|" P68

5.16 Discussion

A **graph** is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects of the graph correspond to vertices (nodes) and the relations between them correspond to edges. A graph is depicted diagrammatically as a set of dots depicting vertices connected by lines or curves depicting edges. Formally,

> "*A graph $G = (V, E)$ consists of V , a non-empty set of vertices (or nodes) and E , a set of edges. Each edge has either one or two vertices associated with it, called its endpoints.*"

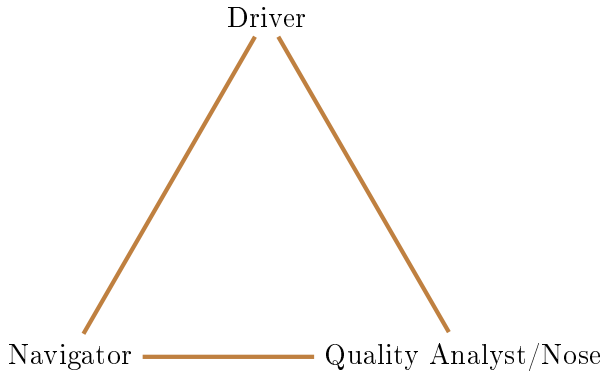
There are several types of graphs distinguished on the basis of edges, their direction, their weight etc. **Complete Graph** is a simple graph of n vertices having exactly one edge between each pair of vertices is called a complete graph. A complete graph of n vertices is denoted by Kn . Total number of edges are n*(n-1)/2 with n vertices in complete graph.

A graph is said to be **planar** if it can be drawn on a plane without any edges crossing. Such a drawing is called a planar representation of the graph. When a **connected graph** can be drawn without any edges crossing, it is called **planar**. When a planar graph is drawn in this way, it divides the plane into regions called **faces**.

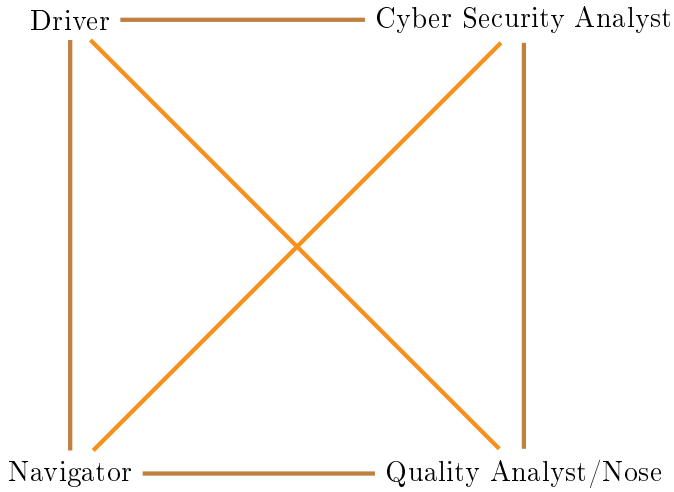
Euler's Theorem. *There is a connection between the number of vertices (v), the number of edges (e) and the number of faces (f) in any connected planar graph. This relationship is called Euler's formula. Euler's Formula for Planar Graphs is for any connected planar graph with v vertices, e edges and f faces, thus,* $v - e + f = 2$.

Kuratowski's Theorem. *A graph is planar if and only if it contains no subdivision of K5 or K3,3.*

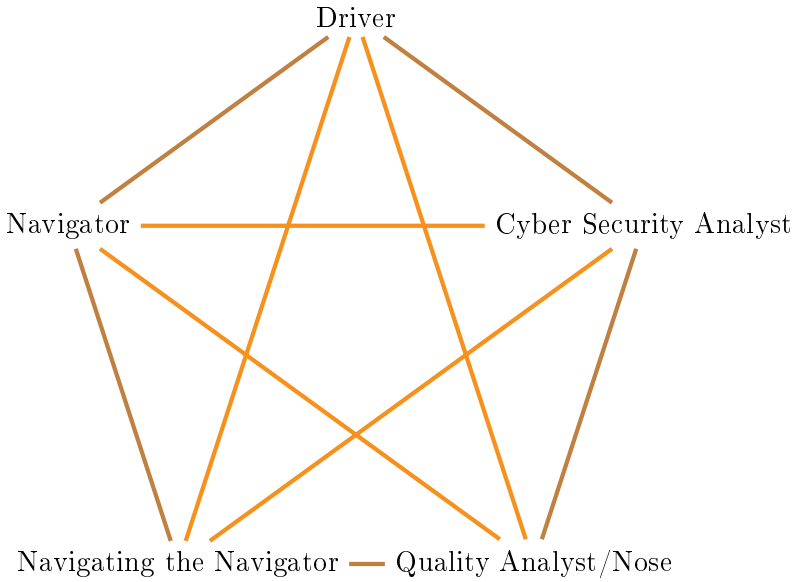
Following is illustrated a graph K3 complete with three edges. Suppose their nodes are the Team Members and the edges are a Pattern of Interaction. Thus, there are three possibilities for Patterns of Interaction.



Following is illustrated a graph K4 complete with six edges. Suppose their nodes are the Team Members and the edges are a Pattern of Interaction. Thus, there are six possibilities for Patterns of Interaction.



Following is illustrated a graph K5 complete with ten edges. Suppose their nodes are the Team Members and the edges are a Pattern of Interaction. Thus, there are ten possibilities for Patterns of Interaction.



Note in 5.6 the Graph is a complete graph planar, so the lines do not cross. However, it is hard for the sound waves to perform this curve.

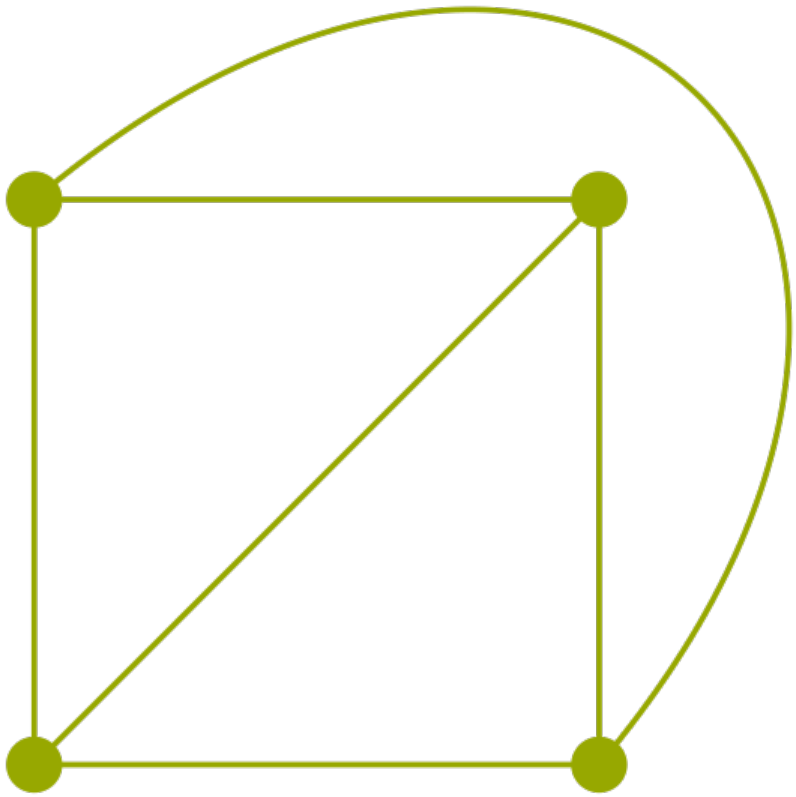


Figure 5.6: *Planar complete graph K4.*

Note in 5.7 the Graph is also a complete graph planar, so the lines do not cross. It is a triangle without crossing any of your edges, and its lines are straight. Thus, it is more natural for sound waves to travel in the air directly with no obstructions. Additionally, the sound waves do not cross between them.

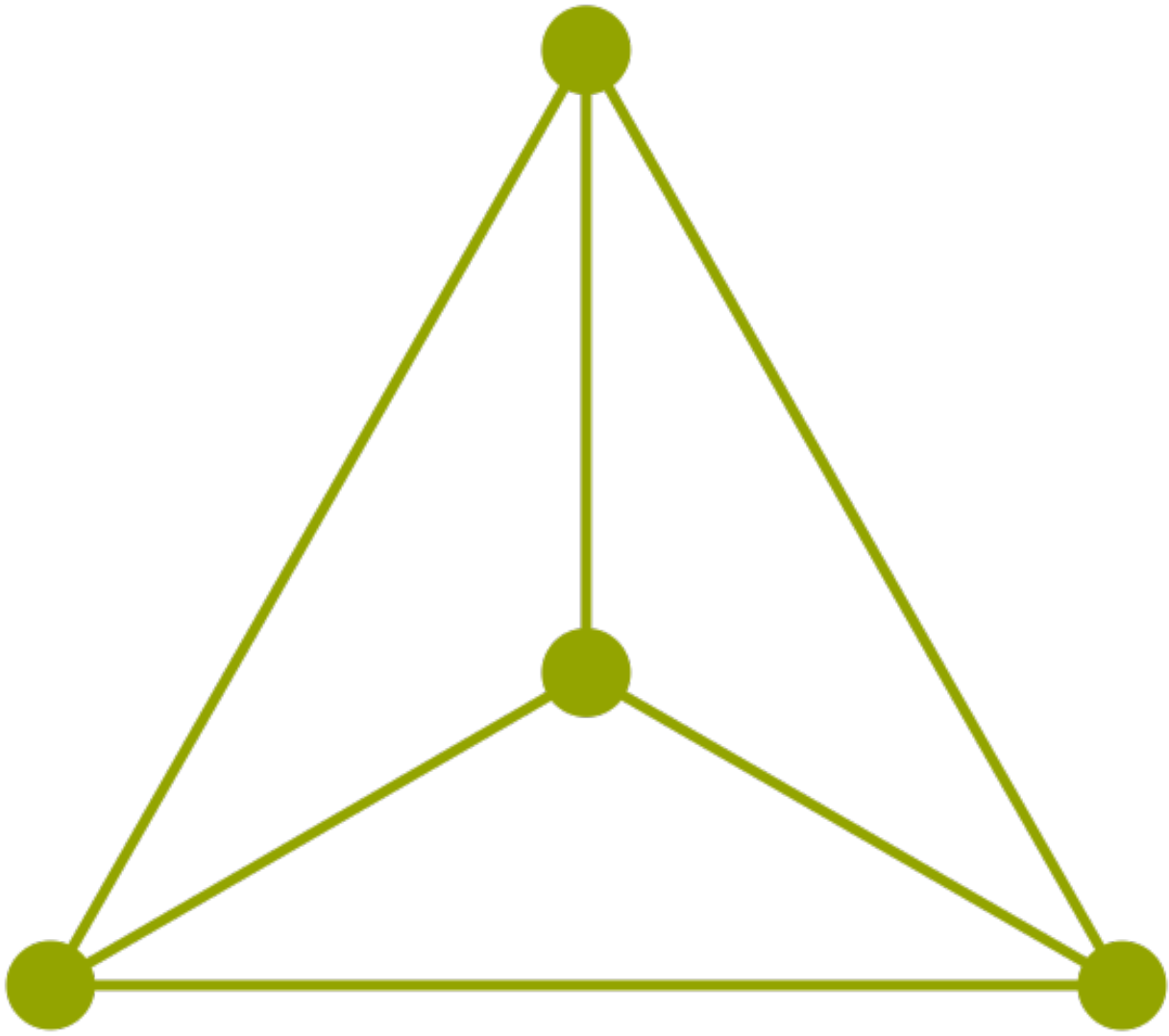


Figure 5.7: *Another possibility of to draw the Planar complete graph K4. It is a triangle that does not cross any of your edges, and its lines are straight.*

5.16.1 Group Size

Note in 5.8 the Graphs are also complete graph planar, so the lines do not cross until K4. But K5 crosses its edges, and its lines are straight. Thus, it is more natural for sound waves to travel in the air directly. However, the sound waves do cross between them in K5. Therefore, the Communication Space in K5 (5 Mob Programmers), considering face-to-face Mob Programming, is more challenging than with 3 or 4 Mob Programmers.

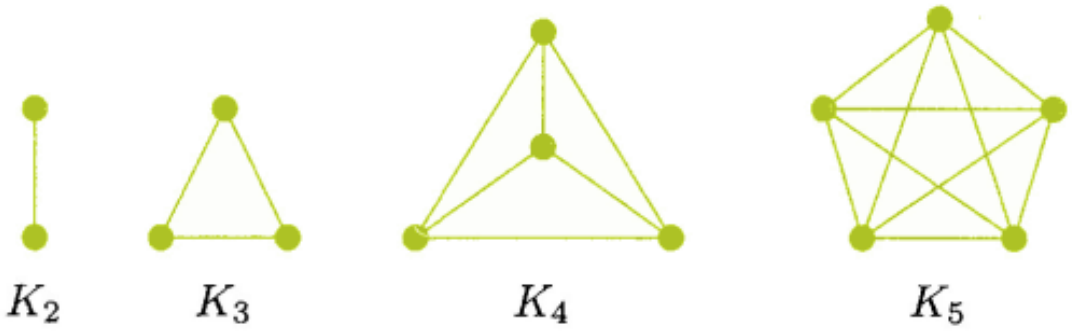


Figure 5.8: *Complete graph K5 is not planar. Note it is impossible to draw K5 without crossing any edges. On the other hand, observe that K2, K3, and K4 are not cross their lines. Therefore, they are planar.*

Groups of size three, four, and five performed better than the best of an equivalent number of individuals, but groups of size two performed at the level of the best of two individuals. Groups of size three, four, and five performed better than groups of size two but did not differ from each other. These results suggest that groups of size three are necessary and sufficient to perform better than the best of an equivalent number of individuals on intellective problems Laughlin *et al.* (2006).

5.17 Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable

This section presents and describes the gathered results of the GT study conducted about Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors, in the form of our Six C's model of Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, elaborated in terms of the Six C's theoretical model for the category Structure Breaks.

The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;
2. **Conditions**: Factors that is prerequisite for having Structure Breaks is: Mental Exhaustion;
3. **Causes**: The reasons are: very complex problem, is hard collaborates all the time, and a long period without an interval;
4. **Consequences**: Outcomes or effects of having Structure Breaks, that is, a contributes positively to Problem-Solving;
5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. Learning something new all together or solo, Mini-retrospectives, Pair Programming, Members Rotation among the Teams, and Pair Programming Simultaneous Style;
6. **Covariances**: Correlations between different categories: Structure Breaks vary with Different Skills Levels, Group Size, Program Experience, Complexity, Bad Day or Personal Problems. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

The Six C's theoretical coding paradigm was used for presenting the results in Figure 5.9. It is SIX C's model of our Six C's model of Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable. Figure 5.9 to illustrate our Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable. The category Structure Breaks is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

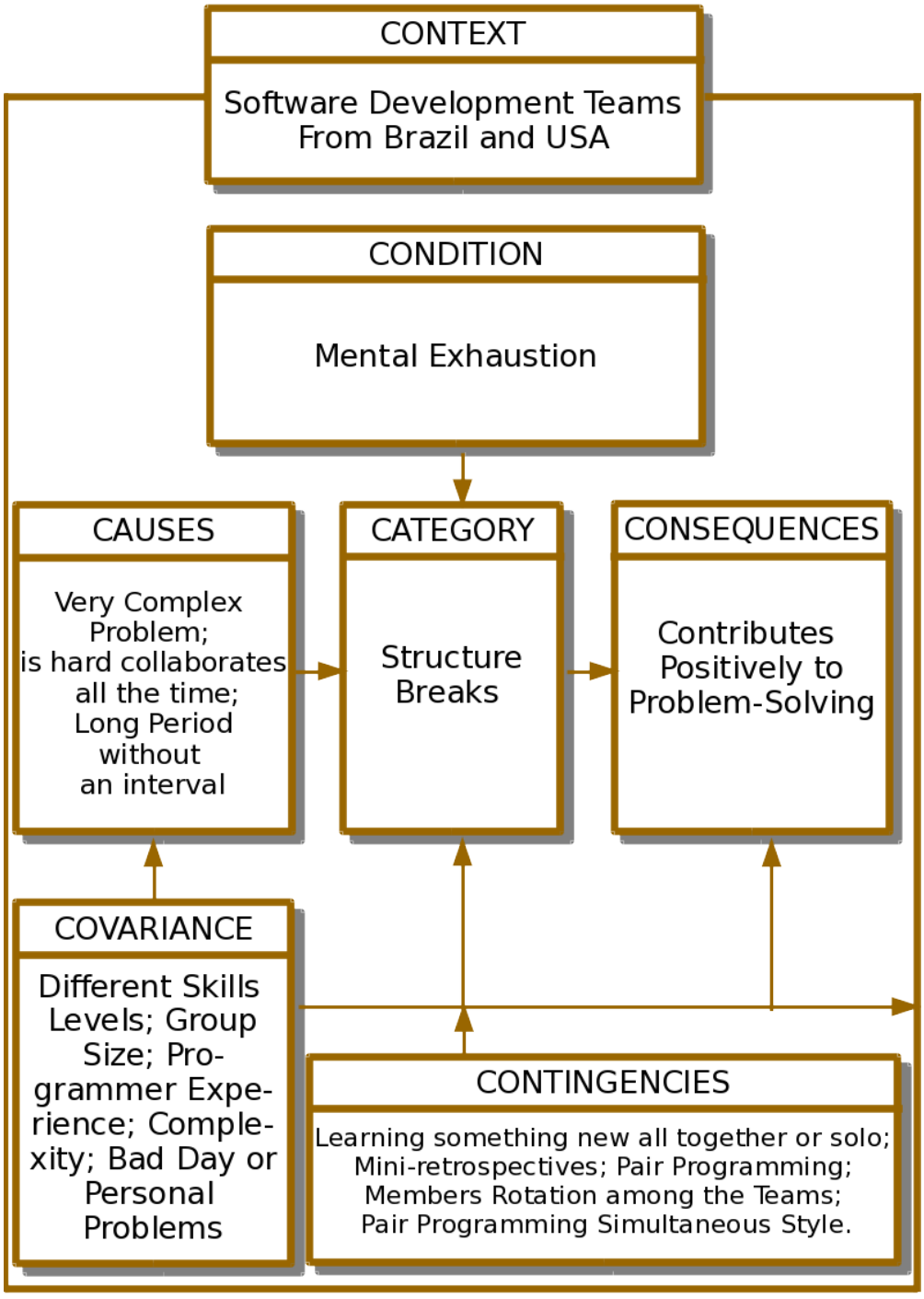


Figure 5.9: *Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable*

5.17.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

5.17.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., there is Mental Exhaustion. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

5.17.3 Causes

Causes define the reasons for the core category to occur. The causes of the need to have a Structure Break are: the team is dealing with a Very Complex Problem, or because is it hard to collaborates all the time, or they are a long period withou an interval.

5.17.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of having a Structure Break is that contributes positively to problem-solving.

5.17.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. Five different contingencies were identified:

- Learning something new together in a group or solo;
- Mini-Retrospectives;
- Pair Programming;
- Members Rotation among the Teams;
- Pair Programming Simultaneous Style;

5.17.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Structure Breaks vary with Different Skills Levels; Group Size; Programmer Experience; Complexity; Bad Day or Personal Problems.

- If appears, a considerable difference in the skills levels of the team members emerges, a need for extra communication to accommodate these differences. Moreover, this differences often leads to false assumptions on how much each of them can contribute, or frustrations because of difference in pace;
- Else if the Group Size trade-off with the Complexity was unbalanced, e.g., not ideal Group Size for the Complexity of the problem, it could be more stressful for the developers rather than an ideal Group Size for the respective Complexity of the problem;

- Else case, the programmers with less experience need to learn more than more experiments, and this effort of leadership could be Exhausting;
- Otherwise, there are many possibilities depending on the different personality profiles of the developers in the team. Moreover, bad days sometimes could happen or if some developer has a personal problem affecting their performance. It is needed to take care of the personal problem of one Mob Programmer, which does not affect the entire team for more time than needed.

5.18 Conclusion

Structure Breaks are fundamental and fruitful in managing complexity and preventing Mental Exhaustion. If a Mob Programming session has one person more experience on the topic: Do not assume they know best. Maybe the need to explain why they are doing things the way they are will bring them new insights. Asking questions on the how and why can lead to fruitful discussions and better solutions. If a Mob Programming session has one person less experience on the topic: Do not assume they cannot contribute much to the solution. You might be stuck wearing blinders and a different viewpoint can help you to come to a better solution. Also, remember that having to explain a concept is a great opportunity to test if all team members really understood it and thought it all the way through.

A conversation at the beginning of the Mob Programming session can help you to understand differences between your styles, and plan to adapt to that. Start your first session with questions like "How do we want to work together?", "How do you prefer to Mob Programming?". Be aware of how you like to work and how you are efficient, but also do not be closed off to other approaches and permit yourself to discover something new.

At the end of a day of Mob Programming session, do a round of feedback for each other. If the idea of giving feedback seems daunting to you, think about it more as a mini-retrospective. Reflect on how you both felt during the Mob Programming session. Were you alert? Were you tired? What made you feel comfortable, what not? Did you switch the keyboard often enough? Did you achieve your goals? Is there anything you would like to try next time? It's good to make this a routine early on, so you have practice in giving feedback when something goes wrong.

Face the challenges as a team and do not leave conflicts to individuals. You can do this for example by organising a session on Mob Programming in which you discuss how to deal with difficulties together. Start the session by collecting the benefits of Mob Programming, so that you know what you all want to get out of it. Afterwards collect the challenges each individual feels when Mob Programming. Now the group can think about which actions might help to improve. You could also collect the hot button triggers of the team members: What makes you immediately feel uncomfortable when Mob Programming?

Perhaps, do not Mob Programming 8 hours a day, agree on core coding hours with your team and keep it to a maximum of 6 hours per day. Maybe you also want to allocate a few hours self learning time.

When a Mob Programmer feels that they do not have the collective knowledge to approach a problem, split up to read up and share back, then continue implementation.

"|*There were moments of mental exhaustion even in mob programming sessions, although some members reported that they found programming alone more exhausting.*|" P66

"|*Intervals between sessions and conversations with colleagues helped to reduce mental exhaustion.*|" P66

"*Yes, it is a lot more exhausting than regular work. We took 15-20 minutes breaks together, and we also had occasions where a person would step back and handle something to, later on, return to the mob.*" P15

"| *As negative points, I mention the team's fatigue after the end and the reduced speed, notably due to the pilot rotation.* |" P189

"*Positives: Quality of code produced. Knowledge shared. Negatives: Tend to "postpone" documentation. Drains more energy from the engineers (not sustainable if done 8 hours per day every day).*" P193

"On our remote mob, the facilitator can also play a great role in live-Slacking (or whatever is your insta-message tool of choice) the intent, activity and decisions of the mob. By communicating like this, other team members who join the mob later can more easily pick up the context, or even offer suggestions or advice from afar, without interrupting the flow." P47

Chapter 6

Challenges to Start Adopting Mob Programming

This chapter presents the core of our grounded theory of the Challenges faced for starting the adoption of Mob Programming. These categories are grounded in the data observed, questionnaires applied, and interviews done.

The quotations are presented verbatim from the interview transcripts or the questionnaire answers. When they are with square brackets symbol "[ ]" is used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Ellipsis, a punctuation mark "..." used within quotations to indicate that something has been left out, i.e., to indicate an omission from speech or writing of a word or words that are unnecessary to the specific point or able to be understood from of the context presented, the forfeit of one or more items from a construction to avoid repeating the identical or equivalent items that are in a preceding or following making. While double three full stops preceded and followed by spaces " ... ... " indicate combining two sentences referring to the same context but derived from different parts of the same interview or questionnaire. The quotation between pipes, as known as vertical lines, or vertical slash, "| |" means a translation from Portuguese occurred in the sentence.

6.1 Convincing managers and co-workers

Advocates of Mob Programming, including the author of this research, often struggle to convince their senior managers or other programmers to make Mob Programming part of a daily work routine in the team.

Persuading others about the efficacy of Mob Programming with facts, data, and related research could help. I faithfully hope this text helps in this satisfy. However, a key element should always be everybody has the same understanding, e.g., by reading an article about Mob Programming or beginning with a presentation about Mob Programming. Then find a way to try it out, either by starting with one Mob Programming session and sharing their experience with the others or reading this chapter about the difficulties other teams faced when starting adopted. Alternatively, by an experiment using Mob Programming, or perhaps try it for a week. A retrospective to share the strengths and weaknesses could help.

We are only starting to study this discovery of Mob Programming, it is not ethical to force a practice on people, and perhaps it does not work for all. A good suggestion might start Mob Programming with only a part of the team, at least in the beginning. From our data analyzed, the best way to convince people is by having regular exposure to the practice and experiencing the benefits and fun their team members have while using Mob Programming.

My Master of Science Dissertation is about a Pair Programming extension, permitting it simultaneously and further pursuing the goal of improving the economic aspects implicated in the practice. And, I did hear in the last twenty-seven years from the Senior Management about paying twice for the same code. The same old common question of senior management that comes up frequently in this situation concerning the economic aspects of the practice: Does Mob Programming just cost much more money, and is it ultimately worth the extra cost because of the increased quality and team benefits? There are no studies on the topic. However, this study of Mob Programming could be cited for providing evidence that it is worthwhile to use it. Software development is a process full of change and uncertainty, with many aspects beyond lines of code, and many factors related to productivity in software development that are hard to compare and measure, e.g., analysis, testing, or quality.

6.2 Keeping the Focus

When the team use mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. It was observed a correlation with the leading challenge of keeping the whole Mob Programmers focused on solving the problem.

> "|*We principally use the roles of Pilot and Navigator. The idea is that all members are always aware of the code quality. Thus, it is expected the produced codes are within the established standards. However, if something is done outside the code standard, the Navigators seek to correct it immediately. If there is doubt, some navigators seek to do research in parallel with what is being at the moment developed. I believe that, although the pilot has the keyboard, this limits him a little in terms of the logical development of the system since he has to worry about writing the code itself. Navigators, on the other hand, have a wide range of possibilities for action, having more freedom of discussion, research, and exchange of ideas.*|" P66

Following is the team answer for the question "What were the difficulties when starting to use Mob Programming?". This is the same team that use mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined.

> "|*The main difficulty is keeping the entire group focused on solving the problem.*|" P66

The other Mob Programming teams used more team roles than only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Additionally, these other teams reported distinct difficulties in beginning with Mob Programming when answering the question, "What were the difficulties when starting to use Mob Programming?". Moreover, none of these other groups claims about the problem of the difficulty of keeping the entire group focused on solving the problem.

Toward being disciplined with the thesis structure. We detailed in the chapter "Literature Review" the literature on Pair Programming, and in the session 2.2.5 "Mechanisms to improve the performance of pair programming" that argues some behaviors to Navigator toward the Pair Programming session be productive and not lose the focus. What corroborates with our thesis about well-defined roles for all team members helps everyone maintain the focus.

6.3 Different skill levels

When people with different experience levels are together in the same Mob Programming session, this often leads to false assumptions on how much each of them can contribute, or frustrations because of difference in pace. It is more challenger to get a Productive Dialog.

If a Mob Programming session has one person more experience on the topic: Do not assume they know best. Maybe the need to explain why they are doing things the way they are will bring them new insights. Asking questions on the how and why can lead to fruitful discussions and better solutions.

If a Mob Programming session has one person less experience on the topic: Do not assume they cannot contribute much to the solution. You might be stuck wearing blinders and a different viewpoint can help you to come to a better solution. Also, remember that having to explain a concept is a great opportunity to test if all team members really understood it and thought it all the way through.

> "*In the progress of innovative and complex systems, learning must be an integral part of daily work activity. Mob Programming provides an ideal environment for this. By watching others, team members learn in real-time how to make better use of the tools and get their day-to-day work done. More importantly, they will soon gain a deeper understanding of the business rules and problem domain and consequently improve their problem-solving skills as they observe and question others about proposed solutions. 1. Do you agree with the above statement? yes 2. Please, describe if Mob Programming helped in understanding the problem domain. Based on your team's experience, if your team observed an improvement in technical competence. Yes, since we seek to include people in the mob who are experienced in the domain and tech, often these people help others grow in knowledge in these things as well.*" P197

6.4 Mental Exhaustion

Taking a break could help in moments of Mental Exhaustion. Have a mini-retrospective on the issue to see what we can do to help.

> " *Every programmer has gone through moments of mental exhaustion when it can take a long time to solve even the most trivial problem. Most programmers are familiar with this issue, including those who do not practice Pair Programming or Mob Programming. It is common to invite a fellow programmer to take a look at their code in these moments of exhaustion or will seek to talk, for example, during a break taking coffee with another programmer. In a Mob Programming session, these moments are practically non-existent. 1. Does your team agree with the above statement? Mostly Was it unanimous? Mostly Did Mob Programming help you avoid at least a little bit to handle these moments of mental exhaustion? Yes! 2. Have there been moments of mental exhaustion in your team? Sometimes How did you cope? Take a break. Have a mini retrospective on the issue to see what we can do to help.* " P196

> "*1. Does your team agree with the above statement? Was it unanimous? Did Mob Programming help you avoid at least a little bit to handle these moments of mental exhaustion? Yes, in our team even when not pairing/mob programming engineers seek each other for help/review/feedback and input. I'm not sure mob programming avoid those, the difference is that they happen less frequently while mobbing. 2. Have there been moments of mental exhaustion in your team? How did you cope? What did you think of the way you guys handled it? Yes. Switching tasks/re-arranging the staff/teams. Often the engineer brings up and asks for help, which I find adequate.*" P42

> "|*There were moments of mental exhaustion even in mob programming sessions, although some members reported that they found programming alone more exhausting.*|" P66

"|*Intervals between sessions and conversations with colleagues helped to reduce mental exhaustion.*|" P66

"*Yes, it is a lot more exhausting than regular work. We took 15-20 minutes breaks together, and we also had occasions where a person would step back and handle something to, later on, return to the mob.*" P15

6.5 Intense collaboration can be hard

Working so closely with another person for long stretches of time is intense. You need to communicate constantly and it requires empathy and interpersonal skills.

"| *As negative points, I mention the team's fatigue after the end and the reduced speed, notably due to the pilot rotation.* |" P189

"*Positives: Quality of code produced. Knowledge shared. Negatives: Tend to "postpone" documentation. Drains more energy from the engineers (not sustainable if done 8 hours per day every day).*" P193

de Aguiar (2003) propose a minimalist approach for documentation as a whole, is cohesive, and represents a possible solution for the problem of documentation in a Mob Programming session. Although minimalism is not very tangible because only a few characteristics are readily visible, the majority is dissolved in design decisions, e.g., the selection of Wiki as the principal documentation environment is brief and allows writing and reading in any order.

You might have differences in techniques, knowledge, skills, extraversion, personalities, or approaches to problem-solving. Some combinations of those might not match well and give you a rocky start. In that case, you need to invest some time to improve collaboration, and make it a mutual learning experience instead of a struggle.

A conversation at the beginning of the Mob Programming session can help you to understand differences between your styles, and plan to adapt to that. Start your first session with questions like "How do we want to work together?", "How do you prefer to Mob Programming?". Be aware of how you like to work and how you are efficient, but also do not be closed off to other approaches and permit yourself to discover something new.

At the end of a day of Mob Programming session, do a round of feedback for each other. If the idea of giving feedback seems daunting to you, think about it more as a mini retrospective. Reflect on how you both felt during the Mob Programming session. Were you alert? Were you tired? What made you feel comfortable, what not? Did you switch the keyboard often enough? Did you achieve your goals? Is there anything you would like to try next time? It's good to make this a routine early on, so you have practice in giving feedback when something goes wrong.

There are excellent trainings and books that can help you deal with interpersonal conflicts and difficulties, for example on difficult conversations.

Face the challenges as a team and do not leave conflicts to individuals. You can do this for example by organising a session on Mob Programming in which you discuss how to deal with difficulties together. Start the session by collecting the benefits of Mob Programming, so that you know what you all want to get out of it. Afterwards collect the challenges each individual feels when Mob Programming. Now the group can think about which actions might help to improve. You could also collect the hot button triggers of the team members: What makes you immediately feel uncomfortable when Mob Programming?

6.6 Interruptions by meetings

A agenda full of meetings probably happens in every team. Meetings are necessary to discuss, plan and agree on things you are going to build, but on the other hand they interrupt the flow. When a team practices Mob Programming the effect of too many meetings can get even worse. If each of the persons in Mob Programming session has meetings at different times, the interruptions are multiplied.

One approach is to limit the time slots in which meetings happen, for example by defining core Mob Programming hours without meetings, or by blocking out no-meeting-times with rules.

It is also worth thinking about meeting length and overall amount. Which meetings do you really need? What goals do they have and how can you improve their quality, for example with proper preparation, facilitation, and a clear agenda.

Check the team members calendars together at the beginning of the Mob Programming session, make sure that there are enough time to start the Mob Programming session. If you have any meetings consider attending them as a Mob. Rely on your Product Owner, or other esternal team members, to keep interruptions away from the team in the core Mob Programming session.

> "*New Mobbing teams typically have to endure the nature of the interruptions created by people getting used to Mob programming. Examples of these issues are background noise, and side conversations. An interruption that would normally slow down development.*" P195

Consideration from Woody Zuill All team related telephone calls and email communications are done "as a team": We sign our emails as "The Dev Team" and have a group email address. When we make or take phone calls we mention to the other party that there are other team members present: "Hi Mary, this is Woody on speaker phone with a few of the other team members".

6.7 Embracing the Vulnerabilities with Kindness

To participate in a Mob Programming session requires vulnerability. It means sharing all that you know and all that you don't know. This could be hard for developers, because programmers are presured to be smart. According to Brown (2015), it is common because most of us try to fight those feelings, we strive to appear perfect.

> "|*Working in a group of people in a Mob Programming session [ ], it can be hard to show that you don't know something or feel insecure about a decision. Especially in our software industry because there are myths like the '10x engineer' and where we tend to judge each other by how many programming languages dominate (Polyglot Programmer) or what design decisions we took ten years ago.*|" P177

Vulnerability is often connected with weakness and in most modern cultures the display of strength is the norm. However, the Courage is born out of Vulnerability, not strength. This finding of Brene Brown's research on shame and "wholeheartedness" as she has laid out in several talks and books, vulnerability is actually a very important ingredient for innovation and change. Based on twelve years of research, she argues that vulnerability is not weakness, but rather our clearest path to courage, engagement, and meaningful connection Brown (2015).

Brene Brown is a research professor at the University of Houston, where she holds the Huffington Foundation-Brene Brown Endowed Chair at the Graduate College of Social Work. Her books include The Gifts of Imperfection Brown (2010) and Daring greatly: How the courage to be vulnerable transforms the way we live, love, parent, and lead Brown (2015). Her talks include "Vulnerability is the birthplace of innovation, creativity and change" at TED2012 and starts off her speech by

describing the extraordinary impact of giving the first talk in 2010 at TEDxHouston: "The Power of Vulnerability," she mentioned on another occasion a phrase from the song "Hallelujah" from the musician Leonard Cohen, who writes, "Love is not a victory march, it's a cold and it's a broken hallelujah." She says that love is a powerful form of vulnerability, and if you replace the word love with vulnerability in that line, it's just as truthful. Vulnerability is not a victory march, it's a cold, and it's a broken hallelujah. When we're brave with our hearts, when we dare to love and hope, to try, fail, and try again, the result of our daring isn't a victory march as much as it is a quiet sense of liberation mixed with a little battle fatigue.

> *"Embracing our vulnerabilities is risky but not nearly as dangerous as giving up on love and belonging and joy- the experiences that make us most vulnerable. Only when we are brave enough to explore the darkness will we discover the infinite power of our light."* Page 6 from Brown (2010).

Mob Programming requires vulnerability and kindness to help create a safe and trustworthy Communication Space. Moreover, showing vulnerability requires courage, and the challenge is creating a Communication Space where people feel safer showing that they are vulnerable. Again, this is all about embracing the vulnerabilities with kindness, permitting people to trust each other, allowing productive retrospectives, and an environment where people do not feel shame for asking questions.

Being vulnerable is more natural for people on the team who have more authority, either naturally (e.g., because they are well-respected software developers) or institutionally (e.g., because they have a position like "Senior Software Developer" or "Principal Software Engineering" or "Staff Software Engineering" and so on depending of the company). So those people should start being role models, making it the norm and therefore safer for others to be vulnerable with confidence.

6.8 No time for yourself

Mob programmers talked about how being in a constant conversation with each other can be pretty energy draining. Most people also need some time on their own throughout the day. That is especially true for more introverted folks.

When working solo, we quite naturally take time to dig into a topic or learn when we need to. But that can feel like an interruption in Mob Programming. So how can you take that alone and learning time when needed?

> "*Positives: Quality of code produced. Knowledge shared. Negatives: Tend to "postpone" documentation. Drains more energy from the engineers (not sustainable if done 8 hours per day every day).*" P193

Perhaps, do not Mob Programming 8 hours a day, agree on core coding hours with your team and keep it to a maximum of 6 hours per day. Maybe you also want to allocate a few hours self learning time or consider to do a structure break.

> "*When a Mob Programmer feels that they do not have the collective knowledge to approach a problem, we split up to read up and share back, then continue implementation.*" P42

6.9 Discussion

Beck (2000), at the beginning of the first XP Explained book, cites the anthropologist Colin Turnbull in The Forest People Turnbull (2012) and The Mountain People Turnbull (1987), paint contrasting pictures of two societies:

> "*In the mountains, resources were scarce and people were always on the brink of starvation. The culture they evolved was horrific. Mothers abandoned babies to roving packs of feral children as soon as they had any chance of survival. Violence, brutality, and betrayal were the order of the day.* " - First Society

> "*In contrast, the forest had plenty of resources. A person had only to spend half an hour a day providing for their basic needs. The forest culture was the mirror image of the mountain culture. Adults shared in raising children, who were nurtured and loved until they were quite ready to care for themselves. If one person accidentally killed another (deliberate crime was unknown), they were exiled, but they only had to go a little way into the forest, and only for a few months, and even then the other tribespeople brought them gifts of food.*" - Second Society

XP is an experiment in answer to the question, How would you program if you had enough time? Perhaps Mob Programming is a possible alternative in answer to the question: how would you run a software company if you had enough time? However, in the next section a happy surprise, about the possibility of **Mob Programming** contributes positively with the productive factors, including the reduction of the time-to-market. A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors.

6.10 Embracing Vulnerabilities with Kindness Theory

This section presents and describes the gathered results of the GT study conducted about Challenges to Start Adopting Mob Programming, in the form of our Six C's model of Embracing Vulnerabilities with Kindness Theory elaborated in terms of the Six C's theoretical model for the category Kind and Fearlessness Programmer.

The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;

2. **Conditions**: Factors that are prerequisites for having Kind and Fearlessness Programmer are: Driver or other Team Role show Vulnerability when Learning something new;

3. **Causes**: The reason is the Programmers decide to adopt the Kind and Fearlessness Programmer behavior, when a driver is new in the programming language or the navigator needs to teach the driver or slowing down speed due to driver limitation;

4. **Consequences**: Outcomes or effects of having Kind and Fearlessness Programmer, that is, a contribution toward a Safe and Innovative Communication Space;

5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. A Structure Break for Learning related issues with the problem being solved. Pair Programming. Simultaneous Style Pair Programming to not delay all the Group;

6. **Covariances**: Correlations between different categories: Kind and Fearlessness Programmer vary with Different Skills Levels; Programmer Experience; Complexity; Other Team Role needing to learn something new. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

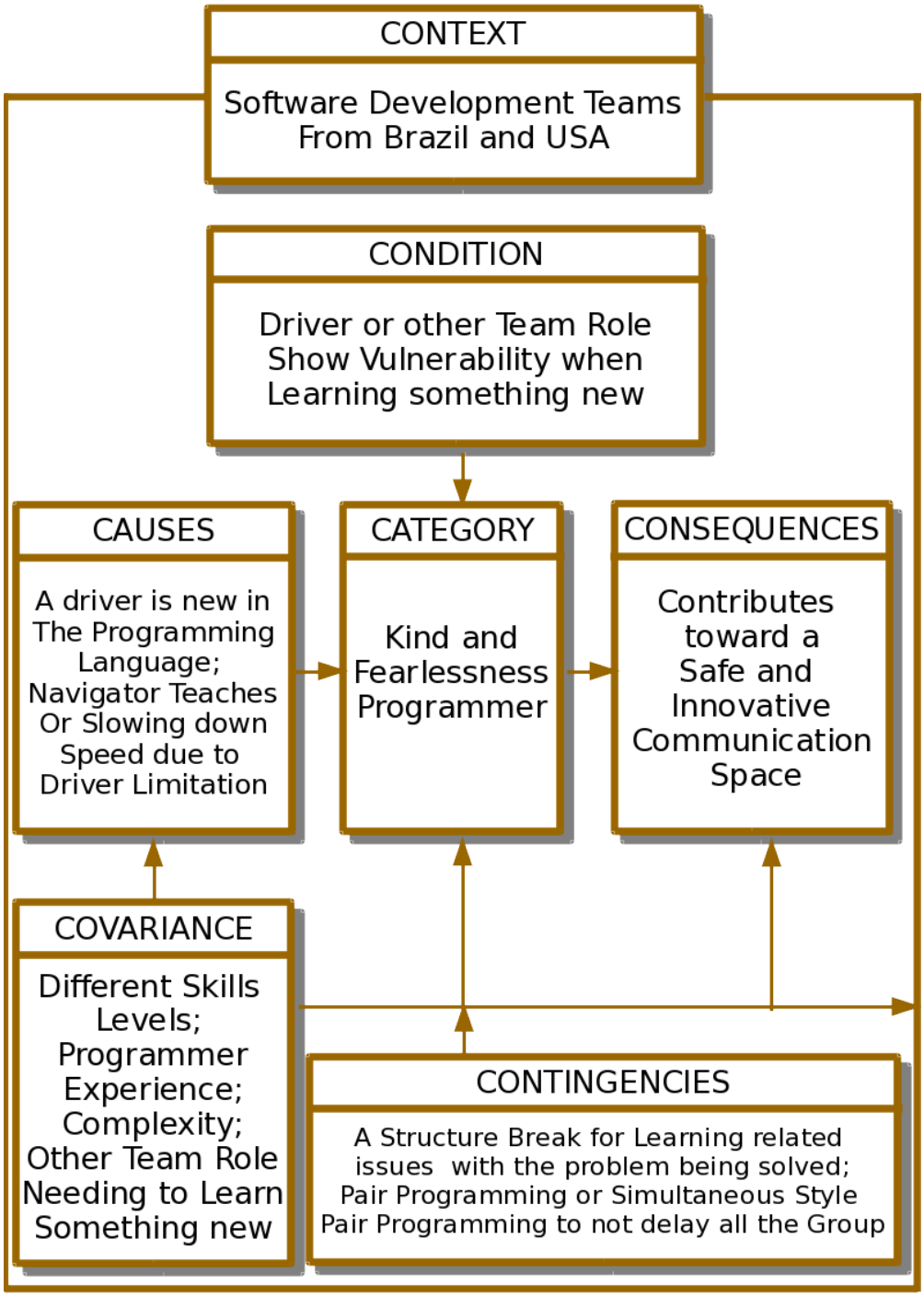


Figure 6.1: *Embracing Vulnerabilities with Kindness Theory*

The Six C's theoretical coding paradigm was used for presenting the results in Figure 6.1. It is SIX C's model of our Six C's model of Embracing Vulnerabilities with Kindness Theory. Figure 6.1 to

illustrate our Embracing Vulnerabilities with Kindness Theory. The category Kind and Fearlessness Programmer is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

6.10.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

6.10.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., Driver or other Team Role show Vulnerability when Learning something new. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

6.10.3 Causes

Causes define the reasons for the core category to occur. The cause to have a Kind and Fearlessness Programmer is a driver is new in the programming language or the navigator needs to teach the driver or slowing down speed due to driver limitation.

6.10.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of having a Kind and Fearlessness Programmer because this behavior contributes toward a Safe and Innovative Communication Space.

6.10.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. Three different contingencies were identified:

- A Structure Break for Learning related issues with the problem being solved;
- Pair Programming;
- Simultaneous Style Pair Programming to not delay all the Group.

6.10.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Kind and Fearlessness Programmer vary with Different Skills Levels; Programmer Experience; Complexity; Other Team Role needing to learn something new.

Chapter 7

Findings Related to Managing the Complexity

This chapter presents the core of our Grounded Theory of the Mob Programming for writing code with high complexity. These categories are grounded in the data observed, questionnaires applied, and interviews done. It is described how Mob Programming could help to Managing the Complexity.

A remarkable analogy made by da Silva and Agusti-Cullell (2008) is creating a self-devouring seamless colossus, when less expected, could implode under the weight of its complexity.

Additionally, Laughlin *et al.* (2006) shows that the collective efforts of groups may be operative in boosting problem-solving skills. In other words, groups perform better than Individuals with the best results at Solving Complex Problems in this study.

The quotations are presented verbatim from the interview transcripts or the questionnaire answers. When they are with square brackets symbol "[ ]" is used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Ellipsis, a punctuation mark "..." used within quotations to indicate that something has been left out, i.e., to indicate an omission from speech or writing of a word or words that are unnecessary to the specific point or able to be understood from of the context presented, the forfeit of one or more items from a construction to avoid repeating the identical or equivalent items that are in a preceding or following making. While double three full stops preceded and followed by spaces " ... ... " indicate combining two sentences referring to the same context but derived from different parts of the same interview or questionnaire. The quotation between pipes, as known as vertical lines, or vertical slash, "| |" means a translation from Portuguese occurred in the sentence.

> "*Continuous Delivery of the work in small steps. Even with very complex systems, make changes with tiny little steps. Each simple step is less likely to harbor something nasty. As faster as you go with small steps in a Continuous Delivery using Mob Programming, we observed less risk.*" P10

7.1 Introduction

Groups of three, four, or five perform better on complex problem solving than the best of an equivalent number of individuals, says Laughlin *et al.* (2006) study appearing in the April issue of the Journal of Personality and Social Psychology, published by the American Psychological Association (APA). This finding may transfer to scientific research teams and classroom problem solving and offer new ways for students to study and improve academic performance, according to the study authors.

In Laughlin *et al.* (2006) study 760 students from the University of Illinois at Urbana- Champaign solved two letters-to-numbers coding problems as individuals or as groups of two, three, four and five people. Previous research has shown that groups perform better than the average individual on a wide range of problems. However, this study tested the relationship between group size and performance as compared to that of an equivalent number of individuals by comparing the number of trials to solutions and answers given for complex problems. The groups of three, four, and five performed better than the best of an equivalent number of individuals on the letters-to-numbers problems.

"We found that groups of size three, four, and five outperformed the best individuals and attribute this performance to the ability of people to work together to generate and adopt correct responses, reject erroneous responses, and effectively process information," said lead author Patrick Laughlin, PhD., of the University of Illinois at Urbana-Champaign.

Moreover, Laughlin *et al.* (2006) shows groups of two performed at the same level as the best of two individuals, suggesting that this group size was too small to introduce the necessary dynamics for optimal problem-solving. However, since groups of three, four, and five were able to achieve the same results, the authors submit that groups of at least three are necessary and sufficient to perform better than the best of an equivalent number of individuals on complex problems that require understanding of verbal, quantitative, or logical conceptual systems. Understanding these systems are necessary skills for scientific research teams, and the finding that groups of three or more perform better than the best of the same number of individuals may indicate that scientific research teams perform better than their top member would perform alone.

7.2 Sharing operational and domain problem knowledge

When you get in a taxi, you need inform the driver where you wanna go if you intend go the location that you need. Agile is about adaptation on changes, not about developer without domain knowledge, the customer always present remember? Developers usually know more about programming and technical tasks than business domain problem knowledge. Thus, developers need learning about operational and business domain problem.

The solution suggest is have a planning phase to collaboratively detect the points at which are need sharing operational and domain problem knowledge. After, work using the Mob Programming.

An example is if in the planning phase is detected some tasks could be useful to share operational and domain problem knowledge, the decision made by the team is using Mob Programming. But, to the other's tasks could use other Programming Technique.

A positive consequence is in the long term this learning of domain problem and operational knowledge help to improve the productivity of the organization. A negative aspect could be the cost to have rooms and projectors adequate to Mob Programming.

> "| *The use of mob programming during the pandemic brought the team closer and expanded the understanding of the project as a whole, facilitating the resolution of bugs and project problems.* |" P83

> "*We seem to have learned and developed more of our technical compentence than our business domain knowledge. It could be that we are working in a very specific business domain that makes it harder for us to learn without being exposed to SMEs and end-users.*" P112

7.3 Quality

Inspection is too late for quality. If you want quality, you need to build it. Software quality is a necessity to be in the development of the software product. A quality assurance analyst's job usually is inspection. Thus, it is too late. Mob Programming builds quality into the product instead of doing a verification.

> "| *The quality has improved due to a better understanding of the problem domain, and because code reviews are done in real-time, the code produced followed all standards and good programming practices. In solo programming, some of these patterns can be ignored or forgotten.* |" P189

> "*Positives: Quality of code produced. Knowledge shared. Negatives: Tend to "postpone" documentation. Drains more energy from the engineers (not sustainable if done 8 hours per day every day).*" P193

7.4 Programmers Satisfaction

Behavioral positive response to the satisfaction of programmers Contributing to managing the complexity.

> "|*Yes, the team has been more productive. Yes, engineers feel like they have become better engineers. Yes, engineers are happy with their personal progress and with the support they are receiving from more experienced engineers in the team.* |" P139

7.5 Programming of complex and critical code

Metaphorically, in determinant situations, the cops call reinforcements. Sometimes, developers need a help to review a code or to develop a complex part of the software system.

The solution is work using the Mob Programming.

> "*Mob Programming makes a lot of sense to me, but I also acknowledge it comes with a cost. It does pay off in the long run for products that aim to be used and reused over several years. For throw-away code like MVPs, or software that will likely be replaced/retired, Mob Programming could also be gold platting which is an anti-pattern.*" P192

A example is in the planning phase was detect the complex and critical parts, facilitating the choice of Programming Technique. Other example is the team chose work using Mob Programming, but unexpectedly happens a critical problem at a complex part of the code of the software, immediately the team decides use Mob Programming and after comeback to work in pairs.

The positive consequence is taking all the advantage of working in pairs and mob programming together. The negative could be the cost to have rooms and projectors adequate to Mob Programming.

7.6 Collective Code Ownership

Collective code ownership abandons any notion of individual ownership of modules. The code base is owned by the entire team and anyone may make changes anywhere. - Martin Fowler

Mob Programmers makes sure that every line of code was touched or seen by at least the major part of the people in the Mob Programming session. Therefor, it helps to manage the complexity. This increases the chances that anyone on the team feels comfortable changing the code almost anywhere. It also makes the codebase more consistent than it would be with single coders only.

> "*Movement between Mobs because they want to avoid developing silos within the overall group. They wanted to make sure that there are at least several people that can handle any particular task that would otherwise be done by a single developer.*" P194

Considerations based on our observations

Mob programming alone does not guarantee you achieve collective code ownership. You need to make sure that you also rotate people through different Mob Programming sessions and areas of the code, to prevent knowledge silos.

7.7 Work in Progress Limit

Limiting work in progress is one of the core principles of Kanban to improve team flow. Having a Work in Progress limit helps your team focus on the most important tasks. Overall team productivity often increases if the team has a Work in Progress limit in place, because multi-tasking is not just inefficient on an individual, but also on the team level. Especially in larger teams, Mob Programming limits the number of things a team can work on in parallel, and therefore increases the overall focus. This will ensure that work constantly flows, and that blockers are addressed immediately.

> "*Automated testing, single item work in progress, and Agile in general.*" P192

Limit your team's Work in Progress to the number of mob programmers on your team and make it visible in your team space (or, if you work remotely, in your online project management tool). Have an eye on the limit before picking up new tasks. Work in Progress limit discipline might naturally contributes to an innovative Communication Space.

7.8 More than two pairs of eyes doing an instant Code review and Refactoring

When using Mob Programming, the team has more eyes on the simple and complex details of the code written. Thus, more errors will get caught on the way instead of after the code is typed.

The refactoring of code contributes to higher code quality. Therefore, Mob Programming instant refactoring helps in the code quality. It is easier to improve code when you have more people besides you because you can discuss approaches or other related issues.

Ask each other questions! Questions are the most powerful tool to understand what you are doing and to come to better solutions. If code is not easy to read and understand for one of you, try a different way that is easier to understand.

> "*1. Do you think Mob Programming was beneficial in complex tasks? yes Why? Yes, since the cognitive load can be shared and unique solutions are derived from a combination of ideas. 2. Has the quality of the software improved? yes Did your team measure? Open bugs How was it calculated? Monitoring, analytics, and reports from users.*" P189

If you feel the need to have more code reviews on mob programmed code, reflect if you can improve your Mob Programming sessions. Perhaps in the next retrospective is a good idea to share this feeling.

7.9 Knowledge Sharing

Having more people work on a piece of the code helps the team spread knowledge on technology and domain on a daily basis and prevents silos of knowledge. Moreover, when two minds understand and discuss a problem, we improve the chances of finding a good solution. Different experiences and perspectives will lead to the consideration of more alternatives.

> "*Positives: Quality of code produced. Knowledge shared. Negatives: Tend to "postpone" documentation. Drains more energy from the engineers (not sustainable if done 8 hours per day every day).*" P193

Do not shy away from a Mob Programming session on tasks when you have no idea about the technology involved, or the domain. If you keep working in the area that you feel most comfortable in, you will miss out on learning new things, and ultimately spreading knowledge in your team.

If you notice that a team member tends to work on the same topics all the time, ask them to mix it up to spread their expertise. It can also help to create a skill matrix with the team's tech and business topics and each person's strengths and gaps in each area. If you put that on a wall in your team area, you can work together on getting a good spread of knowledge.

7.10 Productive Dialogues about Source-Code Quality and Deliver Value

Mob programming forces us to discuss approaches and solutions, instead of only thinking them through in our own head. Saying and explaining things out loud pushes us to reflect if we really have the right understanding, or if we really have a good solution. This not only applies to the code and the technical design, but also to the user story and to the value a story brings.

It requires trust between the Mob Programmers to create an atmosphere in which both of you feel free to ask questions and also speak openly about things you do not understand. That is why building relationships within the team becomes even more important when you adopt Mob Programming. Take time for regular mini-retrospective sessions toward creating an Innovative and Safe Communication Space.

7.11 Keeping focus

It is a lot easier to have a structured approach when there are a group: Each Mob Programmer has to explicitly communicate why you are doing something and where you are heading. When working solo, you can get distracted a lot easier, e.g. by just quickly trying a different approach without thinking it through, and then coming back out of the rabbit hole hours later. When work using Mob Programming a team member can prevent you from going down those rabbit holes and focus on what is important to finish your task or story. You can help each other stay on track.

> "*We have been struggling with different setups around architecture decisions. Our current approach is to break down and simplify and focus on the bits instead of the big picture and the long term.*" P13

Make plans together! Discuss your task at hand and think about which steps you need to make to reach your goal. Put each of the steps on sticky notes (or if remotely, subtasks in your ticket management system), bring them in order and go one by one.

Never forget that Productive Dialog is key. Talk about what you are doing and demand explanations from each other.

7.12 Fast onboarding of new team members

Since Mob Programming facilitates knowledge sharing it can help with the onboarding of new team members. New joiners can get to know the project, the business and the organisation with the help of their Mob Programming. Changes in a team have an impact on the team flow. People just need some time to get to know each other. Mob programming can help to minimize that impact, because it forces people to communicate a lot more than they need when working solo.

> "*We often include Mob Programming as part of our onboarding of new Engineers..*" P61

It is not enough to just put new joiners into a Mob Programming session, and then they are "magically" included and onboarded. Make sure to provide the big picture and broader context before their first Mob Programming session, and reserve some extra time for the onboarding. This will make it easier for them to follow along and contribute during the Mob Programming session, and get the most out of it.

Have an onboarding plan with a list of topics to cover. For some topics you might want to schedule dedicated sessions, for other topics the new team member can take the onboarding plan. If something is covered in the Mob Programming session, you can check it off the list. This way the onboarding progress is visible to everyone on the team.

7.13 More modes of thinking combined

As mentioned when we described the classic driver/navigator style earlier, Mob Programming allows you to have different perspectives on the code. The driver's brain is usually more in "tactical" mode, thinking about the details, and the current line of code. Meanwhile, the navigator's brain can think more strategically, consider the big picture, and park the next steps and ideas on sticky notes. Could one person combine these two modes of thinking? Probably not! Having a tactical and strategic view combined will increase your code quality because it will allow you to pay attention to the details while also having the bigger picture in mind.

Remember to do frequently the Driver Rotation and the roles regularly. This helps you to refresh, not get bored, and to practice both ways of thinking.

> "*The cognitive load can be shared and unique solutions are derived from a combination of ideas.*" P199

As a navigator, avoid leave the details of the coding to the driver - the navigator's role is to take a step back and complement the driver with more tactical mode with medium-term thinking.

7.14 Discussion

7.14.1 Reducing the rate at which developers leave the team

Williams and Kessler Williams and Kessler (2003) *apud* Weinberg Weinberg (1971) 'If a programmer is indispensable, get rid of him as quickly as possible'. Although this contextualization works very well to talk about avoiding build a house of cards, we are talking about an important productivity factor: the problem of if your team has a 'key' person and this person leave the team. The problem is to retain the talented people.

The incentive to the Programmers to adopted the technique of **Mob Programming** ideally needs have the support of the senior management. Sharing and recognizing competencies must be appreciated by the organization culture. Fighting against bad programming practices too. Creating an environment-friendly to the conversation between programmers.

Mob Programming helps to change the culture of the organization to **collaboration** could stimulate the programmers to use it and get motivation to stay working to your company.

A very positive consequence is the retaining of the top talent. In the long term increase of the productivity and in the short term could increase the cost and maybe decrease the productivity are negative consequences.

7.15 Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming

This section presents and describes the gathered results of the GT study conducted about Mob Programming for writing code with high complexity, in the form of our Six C's model of Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming elaborated in terms of the Six C's theoretical model for the category Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists.

The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;

2. **Conditions**: Factors that are prerequisites for having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists are: The team prioritizes Quality and CyberSecurity;

3. **Causes**: The reason is the Programming Programming session have enough Group Size to have the Roles: Qualiy Analyst, Nose, and CyberSecurity Analyst;

4. **Consequences**: Outcomes or effects of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists, that is, closer team; knowledge sharing; manage complexity; higher quality software; adherent to the code standard; clean code and safer; reducing the chance of a damaging cyber incident; maximizing resilience;

5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. Automated Code Review tool, e.g., Code Climate, Mezuro or SonarQube; Automated CyberSecurity Risk Management tool;

6. **Covariances**: Correlations between different categories: Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists vary with Group Size and Priorities. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

The Six C's theoretical coding paradigm was used for presenting the results in Figure 7.1. It is SIX C's model of our Six C's model of Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming. Figure 7.1 to illustrate our Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming. The category Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

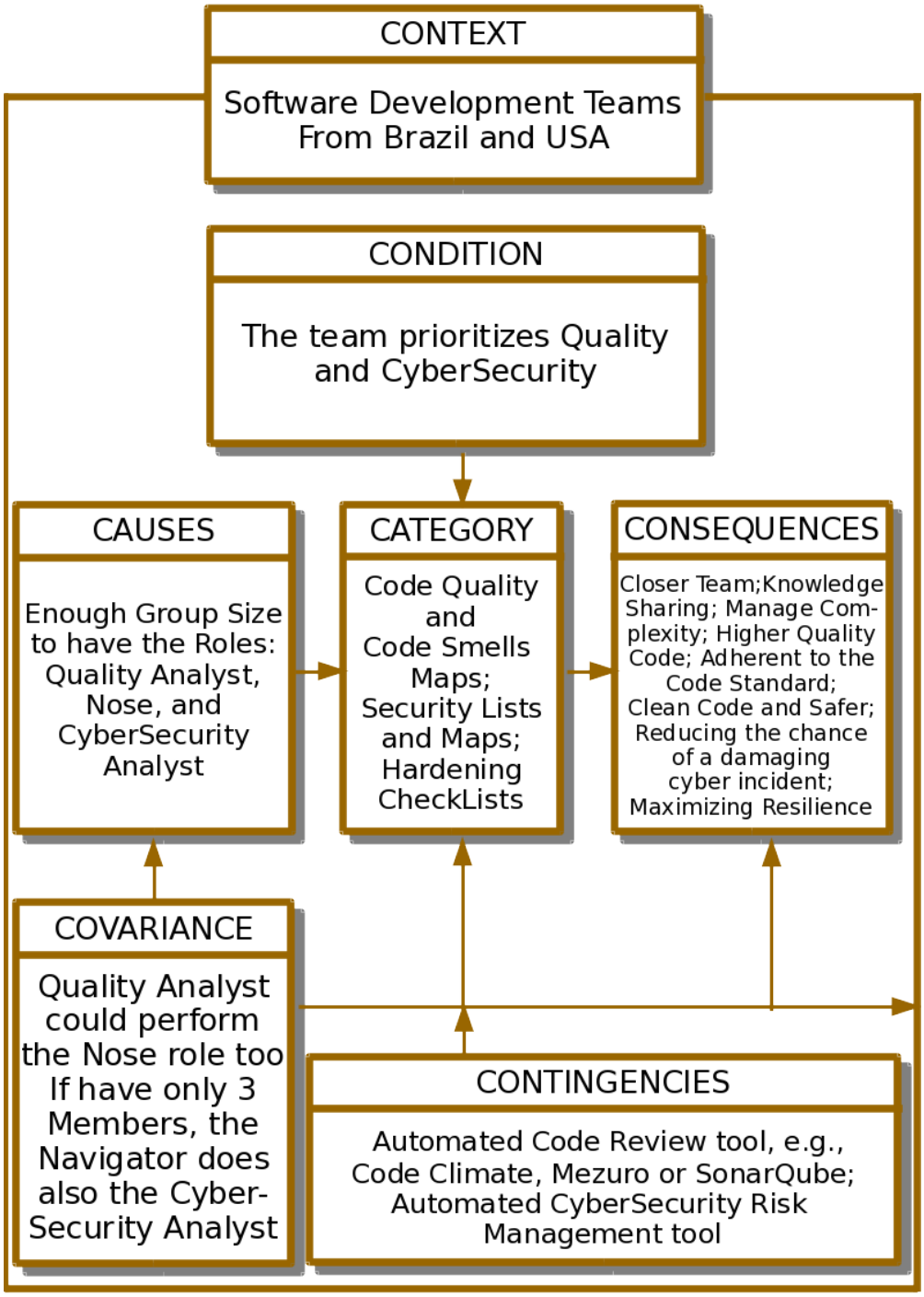


Figure 7.1: *Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming.*

7.15.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

7.15.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., The team prioritizes Quality and CyberSecurity. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

7.15.3 Causes

Causes define the reasons for the core category to occur. The reason is the Programming Programming session have enough Group Size to have the Roles: Qualiy Analyst, Nose, and CyberSecurity Analyst

7.15.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists that contribute to outcomes or goods of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists, these benefits are: closer team; knowledge sharing; manage complexity; higher quality software; adherent to the code standard; clean code and safer; reducing the chance of a damaging cyber incident; maximizing resilience;

7.15.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. Two different contingencies were identified:

- Automated Code Review tool, e.g., Code Climate, Mezuro or SonarQube;
- Automated CyberSecurity Risk Management tool;

7.15.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists vary with Group Size and Priorities.

Chapter 8

Results About Remote Work

This chapter presents the core of our Grounded Theory of the Remote Mob Programming. These categories are grounded in the data observed, questionnaires applied, and interviews done. It is described some differences between Mob Programming and Remote Mob Programming.

The quotations are presented verbatim from the interview transcripts or the questionnaire answers. When they are with square brackets symbol "[ ]" is used to insert missing words to fix grammar or to anonymize participant details (such as names of individuals or companies). Ellipsis, a punctuation mark "..." used within quotations to indicate that something has been left out, i.e., to indicate an omission from speech or writing of a word or words that are unnecessary to the specific point or able to be understood from of the context presented, the forfeit of one or more items from a construction to avoid repeating the identical or equivalent items that are in a preceding or following making. While double three full stops preceded and followed by spaces " ... ... " indicate combining two sentences referring to the same context but derived from different parts of the same interview or questionnaire. The quotation between pipes, as known as vertical lines, or vertical slash, "| |" means a translation from Portuguese occurred in the sentence.

8.1 Introduction

Use Remote Mob Programming effectively is similar to face-to-face Mob Programming and the plurality of other software development processes. The continuous retrospective is a central point in the software development processes. The environment and atmosphere created by Mob Programming contribute to doing retrospectives lightly in an ongoing way behavior because all the team members are together.

The human being is social, so the isolation of a professional working alone at home or not physically present with the others members of the team, is a problem. Open source projects typically are remote, making that a very import problem. Others possible related problems are: avoid the wasting of time due to traffic jams, taking advantage of cheaper developers offshore, experts geographically distributed collaborating with your project, reduction of the time-to-benefit by having programmers working 24 hours developing software in the 7 days a week using remote working, increase the productivity of the team, quality of the home-office work and software produced.

The solution is Mob Programming in a distributed way, where the people are in different physical dependencies and communicated by voice and text messaging software, can be used to increase development productivity and product quality. Nowadays, exist many software able to exchange messages and videos, include with a live option to text, audio, and video. There are too many options to share the screen and work in a group collaboratively.

An example is the programmers wake up, getting ready to the work. Turn on the computer and starts working using software to support mob programming. A negative consequence is considering people are social, the remote working makes people communication not be personal, face-to-face. There are some possible positive consequences: reduction of the time-to-benefit, increase the productivity of the team, quality of the home-office work and software produced.

With Mob Programming, three or more developers work together on the same computer. The aim is to deliver high-quality software as quickly as possible. Such collaboration between a group of ambitious developers is not that easy. If it also has to be done remotely, it seems even more challenging. However, Remote Mob Programming delivers great results.

Change roles regularly as with Mob Programming, Remote Mob Programming always requires one Driver. This person is the only one who is allowed to type and code. An important rule: roles are changed every so often so that each participant has a turn as the driver. That way everyone stays alert.

Create a virtual space. Make sure that everyone on the team can speak to each other at any time, just as when everyone sits at the same desk. So create a virtual space where everyone is present, for example by setting up a 'lobby call' that is on all day. Usually the microphone and webcam are just turned off when there is no communication. When you have a question that everyone should hear, you can turn on your microphone and ask the question out loud. Everyone in the lobby can then respond.

Share your screen. Create small teams of preferably three or four people. There are various tools, such as Microsoft Teams and USE Together, Rocket.Chat, Slack, and so on, that allow you to remotely share your screen and effectively collaborate with several people on a single PC. Moreover, the person who is the Driver can take control of the computer each time.

Turn on the webcam. Turn on your webcam during meetings. If you need to brainstorm or want to discuss something, it is much nicer to be able to see each other. Body language also allows for much better communication. Then you see how your colleagues react to decisions and solutions as they are being made.

Mob Programming is difficult when developers work with different computers and keyboards. Mac and Windows, for example, or azerty and qwerty, or if they work with self-defined shortcuts or with different editors such as IntelliJ or Visual Studio Code.

Useful tools for Mob Programming. There are also tools for making the collaboration a little smoother. For example, Mobster is a very useful tool for Mob Programming. You can enter the names of the participants, after which a driver and a navigator are automatically designated. When the timer goes off the roles are also automatically redistributed. This way you avoid discussions.

8.2 Software setup

The following quotations describe the principal software tools used by the participants.

"*Rocket.Chat, Git, and Visual Studio Code.*" P57

"|*Visual Studio Code and Microsoft Teams.*|" P123

"|*Ruby. Ruby on Rails. Slack. Discord. Tuple. Google hangouts.*|" P79

"|*Microsoft Visual Studio and Slack.*|" P174

"|*Ruby on Rails and plugNmeet.*|" P114

"*I love remote programming daily at work including today. Using AnyDesk tool to share mob station and Teams for audio and video.*" P198

"*Today, using Anydesk tool to share mob station and Teams for audio and video.*" P199

"*We switch to use whatever programming language and tech is needed to solve the problem at hand. We do not limit ourselves to any particular language or technology. The mob decides what to use. Over the last 4 years, I have used the following languages and technologies while mob programming: AngularJs; Angular; TypeScript; Node; Javascript; SQL; SQL Server; MySql in AWS; AWS; AWS DynamoDb; C#; VisualBasic; Python; Bash; Powershell; Cloudformation; Yaml; Xaml; Xamarin Cross Platform Mobile Development; Terraform; Azure DevOps; GoCD; Amplify; Visual Studio; Visual Studio Code; Rider; WebStorm; PyCharm; SQL Server Management Studio; DataGrip; Wallaby; Ncrunch; Chai; Mocha; Jasmine; Flutter; NativeScript; Cucumber; Nunit; MsTest; Aws IoT Core; AWS Lambda; Alexa Skill Development; Etc.*" P199

8.3 Driver rotation techniques

Although Driver rotation techniques, as Facilitators for productive dialogues, are the same as face-to-face Mob Programming. It was observed the Facilitator role has more work when the Mob Programming session is Remote. Driver rotation techniques as Facilitators for productive dialogues. Through an Initial Information System, Timer, Standardized Procedures, Strong-Style, Red/Green Criteria, and a Facilitator.

"On our remote mob, the facilitator can also play a great role in live-Slacking (or whatever is your insta-message tool of choice) the intent, activity and decisions of the mob. By communicating like this, other team members who join the mob later can more easily pick up the context, or even offer suggestions or advice from afar, without interrupting the flow." P47

"| *... The role of the facilitator was played by the agile coach, who conducted the retrospectives and controlled pilot rotation, together with keeping track of time.* |" P189

" *We do often remote mob programming as described here: https://trello.com/c/l96SiQum/254-mob-mentality-on-remote-mob-programming We rotate who is driving and controlling the Anydesk keyboard and mouse for the centralized computer. We often use this timer: http://mobster.cc/ Use tools that make driver rotation easy and seamless. In the spirit of 'just sharing' (https://justsharing.dev/), it has worked well for us to at least start with a short-timer rotation like 4 mins. In my experience, short-timers make it easier for active collaboration from all. Then after trying that, the team can experiment and adapt to something that works better for them.* " P199

" *We do strong style sometimes and it helps us equalize navigation when it is needed. Sometimes we do 'many navigators' (https://jay.bazuzi.com/Mobbing-Pattern-Language/patterns/Many%20Navigators.html) or more freestyle (https://youtu.be/hdzAvcz-_Mo). We try to use whatever style is needed for the problem at hand or what is best for the people in the mob at the time.* " P200

8.4 Retrospectives

Retrospectives help in the continuous improvement process.

> "*Yes. We are constantly reviewing and changing our process. Retrospectives are/were helpful to us. We have been decreasing the frequency because the team grew from 10 to 40. So instead of having one all-hands retrospective every week, we broke down into 4 bi-weekly retrospectives.*" P137

> "*Yes. The team started with weekly and moved into bi-weekly retrospectives.*" P004

> " *Yes, we do retrospectives often and they are very helpful. Some examples/stories from my experience are shared here (https://youtu.be/0_NSt_Y93Q4).* " P199

> " *When my current team is just one small mob, we often do not do any other frequency of retrospective since the mini-retro covers it all. There was not really a need for one.* " P200

> " *Experiments/changes often came out retrospectives. To list all the changes would not be realistic since I have been mob programming full time for 4 years, which means I have been involved in up to a thousand or more mini-retros and up to 200 or more weekly retros. However, I can give some examples: - we changed what IDE we use as a mob since some have more features to automate more for the quality practices, which was helpful. - we changed how we mob such as switching to stronger navigation or moving away from it, which helped to improve collaboration - we changed how we interacted with 3rd party teams to improve flow efficiency Not all changes led to good things and we gladly abandoned those experiments. Not sure if there is an ideal universal retro frequency. In my experience, many small retros throughout the mobbing day is the most beneficial for effective continuous improvement.* " P198

8.5 Communication tools

When using Remote Pair Programming tools allows you to point on the shared screen without interfering with the driver's input via keyboard and mouse.

> "*We first try to communicate verbally without using a pointer of any kind. With remote mob programming, the remote tools allow you to point on the shared screen without interfering with the driver's input via keyboard and mouse. With in person mob programming, we would walk up the screen if needed and point with our finger. Some people used pointing devices sometimes.*" P200

8.6 Automation

The automation of work to avoid downtime, it is different to waste a minute of a team member's time and lose a minute of the entire team.

" *First thing that comes to my mind is build time. We have been actively working on improving the build time to decrease the burden of having to wait for it, specially when we are pairing or mob programming.* " P19

" *Yes, we have automated: Testing, Delivery, Language Translation, Infrastructure, Ticket System manipulation, HR Overhead and 'Paperwork' Processes, and Status Reporting. Almost all have been tremendously helpful except some forms of status reporting automation. I believe what has been helpful and what has not been helpful lines up well with this picture here: https://trello.com/c/1hCsXx6c/159-is-it-worth-automating-chart* " P198

8.7 Structure breaks

Full involvement all the time can be exhausting, especially on complex tasks. Thus, structure breaks are helpful.

"*I believe that creating "slack" for people to think and the process sometimes brings up that breakthrough idea that solves or simplifies the whole solution. I haven't quantified/documented the [exact number] of times this happened with our teams after a break, but this happened indeed, whether it is this you want to know and it is trying to discover.*" P137

" *1. Is full involvement at all times exhausting? Sometimes Did you take breaks together? Yep How went? Excellent, it seems very healthy and I come back to the mob very refreshed 2. Did your team make a break every hour? Yes, typically every 50 minutes there was a 5-10 minute where people could not be behind the screen. What do you think? Sounds great 3. Did intervals help in solving a complex problem? Could they have helped? Yes, for the reasons discussed in this video: https://youtu.be/SW736CjNMjY* " P199

" *We took 15-20 minutes break together and we also had occasions where a person would step back and handle something to later on return to the mob.* " P49

8.8 How many programmers in a session of Mob Programming

No ideal number but whatever is needed for the problem at hand.

"*I believe remotely is harder and makes sense to work with less people if possible. When we are remote we lose a lot of communication cues. In person: 7-8 is ok. Remote: 4-5 is ok.*" P32

"*5-6.*" P112

" *I like to follow the principles of 'Get the whole system in the room' and 'Everyone who is needed is there.' So there is no ideal number but whatever is needed for the problem at hand.* " P199

8.9 Remove one person and team performance

The code and value kept flowing even if someone was absent.

> "| *During one of the Mob Programming sessions, one of the developers had connection problems and unable to be present at the Mob. This did not interfere with the performance of other people on the team.* |" P189

> "*Yes, it has happened in the past many times and it just happened right now. Performance was great just now. The code and value kept flowing.*" P189

8.10 Programmer Satisfaction

Fun is fundamental in life since it fosters interaction and learning Vieira and da Silva (2017).

> "|*Most of our developers were satisfied with remote Mob Programming. But, it is not unanimous.* |" P80

> "*could write a multi page essay on this but the short version is that positives include improved flow efficiency and joy for me. A con is that it amplifies the intensity of issues, which helps to identify issue to work through them, but also makes it more challenging in the short term.*" P199

> "*For me, one of my favorite experiences with mob programming is my first day of mob programming. I had a great time learning from new friends, contributing to the code/solution, and we pushed the solution to production on the first day. There was no waiting for others (like I was used to in previous contexts).*" P200

8.11 Benefits in long term

People in the mob who are experienced in the domain and tech often these people help others grow in knowledge in these things as well. This is an advantage in the long term.

> "*Great benefits, also in the long term. Mob Programming and Remote Mob Programming may seem very complicated and inefficient at first because so many people are involved. But this approach offers great benefits. To begin with, you get to know your fellow developers better, which gives the collaboration a boost in the longer term. Moreover, you can often come up with better and more elegant solutions to complex problems by putting your heads together. Finally, mobbing also ensures that the entire team is aware of all decisions, the code base and functionalities.*" P39

8.12 A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage

This section presents and describes the gathered results of the GT study conducted about Remote Mob Programming, in the form of our Six C's model of A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors elaborated in terms of the Six C's theoretical model for the category Remote Mob Programming.

The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;
2. **Conditions**: Factors that are prerequisites for having Remote Mob Programming are: pressures of time-to-market or cost;
3. **Causes**: a group size of three or variable depending on the complexity; dependency queue reduction due to getting information faster; previous theories satisfied;
4. **Consequences**: Outcomes or effects of having Remote Mob Programming, that is, improve productivity factors in complex tasks; reduce the time-to-market;
5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. Pair Programming Simultaneous Style;
6. **Covariances**: Correlations between different categories: Remote Mob Programming vary with Complexity; Programming Experience; Group Size; Priorities; Team Roles adoption; Absents; innovative Communication Space; Information Flow. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

The Six C's theoretical coding paradigm was used for presenting the results in Figure 8.1. It is SIX C's model of our Six C's model of A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors.

Figure 8.1 to illustrate our A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors. The category Remote Mob Programming is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

Figure 8.1 is a theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors.

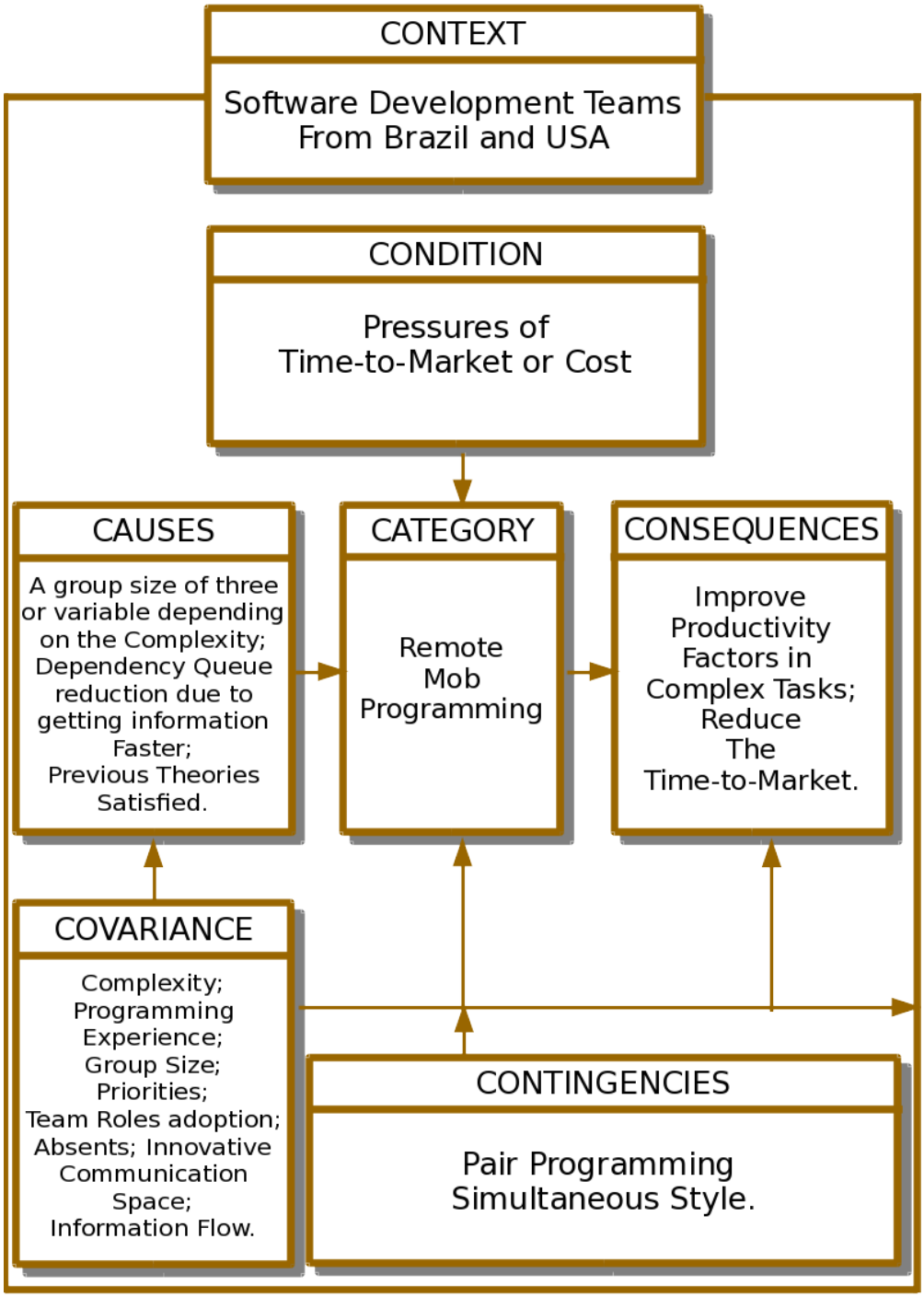


Figure 8.1: *A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors.*

8.12.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

8.12.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., pressures of time-to-market or cost. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

8.12.3 Causes

Causes define the reasons for the core category to occur. The cause to have a group size of three or variable depending on the complexity; dependency queue reduction due to getting information faster; previous theories satisfied.

8.12.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of having a Remote Mob Programming is that improve productivity factors in complex tasks; reduce the time-to-market.

8.12.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. One different contingencies was identified: Pair Programming Simultaneous Style.

8.12.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Remote Mob Programming vary with Complexity; Programming Experience; Group Size; Priorities; Team Roles adoption; Absents; Innovative Communication Space; Information Flow.

8.13 Theory of Benefits of Remote Mob Programming

This section presents and describes the gathered results of the GT study conducted about Remote Mob Programming, in the form of our Six C's model of Theory of Benefits of Remote Mob Programming elaborated in terms of the Six C's theoretical model for the category Remote Mob Programming. The Six C's consist of the following instances: context, conditions, causes, consequences, contingencies, and covariance. Thus, using the Six C's model, we described:

1. **Contexts**: The ambiance, that is, the context of the Mob Programming teams in Brazil and USA;

2. **Conditions**: Factors that are prerequisites for having Remote Mob Programming are: different geographic locations; COVID-19 Pandemic;

3. **Causes**: internet, software, hardware, internet connection, cybersecurity protection, all with good quality; previous theories satisfied;

4. **Consequences**: Outcomes or effects of having Remote Mob Programming are that the Mob Programming benefits are preserved; the possibility of working from anywhere; family and pets closer; little-to-no-commute; greater inclusivity; non-stop working in different time zones;

5. **Contingencies**: Moderating factors between causes and consequences. We have the following possibilities. Remote Pair Programming and Remote Pair Programming Simultaneous Style;

6. **Covariances**: Correlations between different categories: Remote Mob Programming vary with Complexity; Programming Experience; Group Size; Priorities; Team Roles adoption; Absents. Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another.

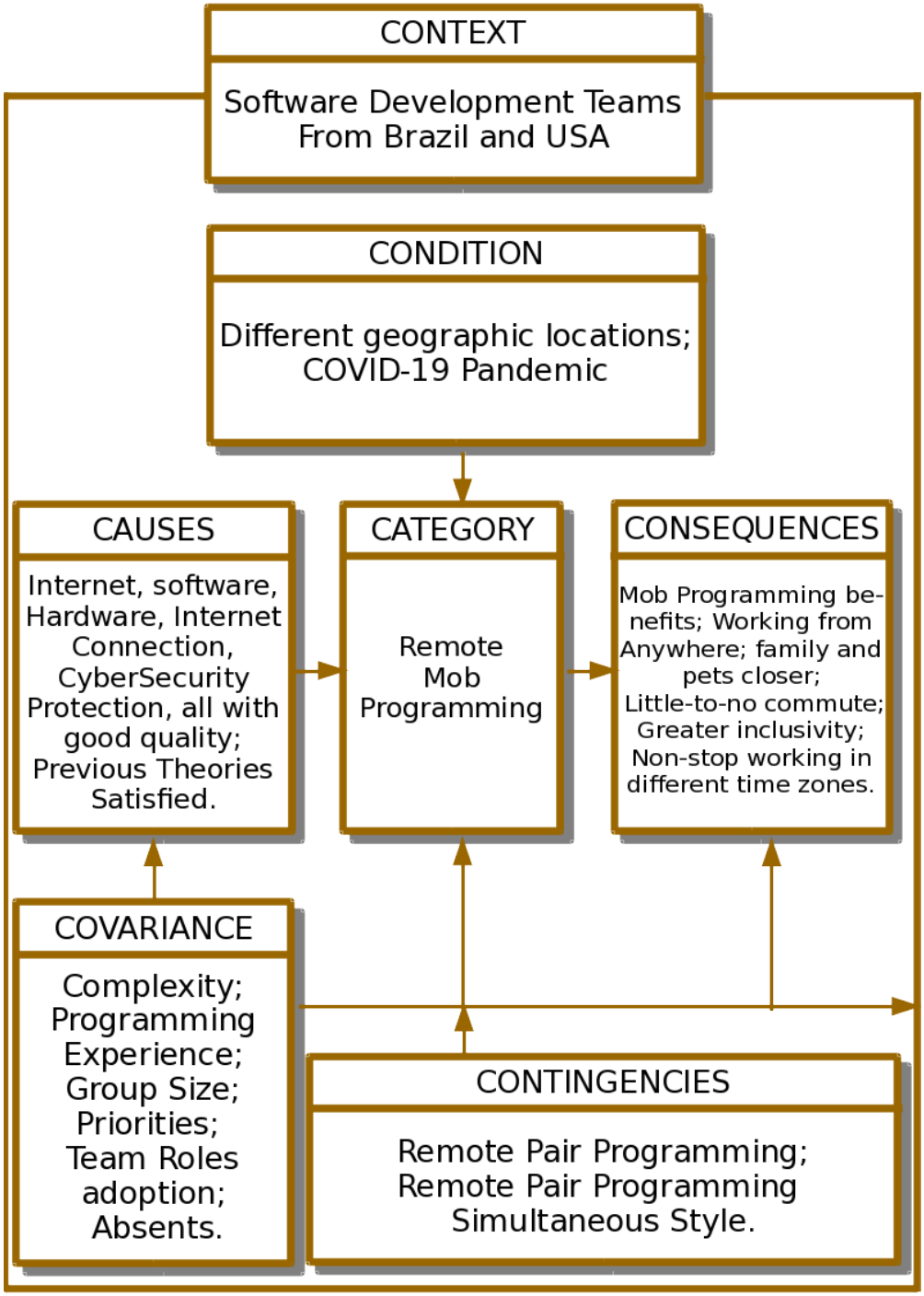


Figure 8.2: *Theory of Benefits of Remote Mob Programming*

The Six C's theoretical coding paradigm was used for presenting the results in Figure 8.2. It is SIX C's model of our Six C's model of Theory of Benefits of Remote Mob Programming. Figure 8.2 to illustrate our Theory of Benefits of Remote Mob Programming. The category Remote Mob Programming is at the center of the diagram. The Six C's are represented in the other rectangles about the centric category. In the following subsections, we have selected quotations drawn from our interview transcript or questionnaire answers that shed particular light on the concepts, i.e., in the following subsections, we describe our theory.

8.13.1 Context

Context describes the setting when the core category occurs. In this research, the context of the Mob Programming teams in Brazil and USA.

Toward being disciplined with the thesis structure. We detailed in the chapter "Introduction" the general introduction to this research, including the context in the session 1.4 "Research Strategy and Research Context" that describes the projects, institutes, organizations, and companies profiles, toward to better contextualization of participants in this study.

8.13.2 Conditions

Conditions define the requisites for the core category to emerge, i.e., different geographic locations; COVID-19 Pandemic. The following sections present, Causes, Consequences, Contingencies, and Covariances of our 6C's model.

8.13.3 Causes

Causes define the reasons for the core category to occur. The cause to work with Remote Mob Programming is internet, software, hardware, internet connection, cybersecurity protection, all with good quality; previous theories satisfied.

8.13.4 Consequences

Consequences define the effects of the occurrence of the core category. We identified the positive effect of Mob Programming benefits. Outcomes or effects of having Remote Mob Programming are that the Mob Programming benefits are preserved; the possibility of working from anywhere; family and pets closer; little-to-no-commute; greater inclusivity; non-stop working in different time zones.

8.13.5 Contingencies

The main goal of the enlisted contingencies is to moderate the causes influencing the core category and the consequences arising from the core category. Two different contingencies were identified:

- Remote Pair Programming;
- Remote Pair Programming Simultaneous Style;

8.13.6 Covariances

Covariance occurs, according to Glaser (1978) and Kan and Parry (2004), when one category changes, with the changes in another. The correlations between different categories: Remote Mob Programming vary with Complexity; Programming Experience; Group Size; Priorities; Team Roles adoption; Absents.

Chapter 9

Conclusion

This chapter summarizes the grounded theory about Mob Programming, the main contribution of this thesis. The next sessions critique our grounded theory and it is a discussion related with the existent literature of remote work and manage to complexity, implications for practice, and suggestions for future work.

The next sections critique our grounded theory and identify the limitations of this study. This is followed by a discussion of the theory in the light of existing literature, implications for practice, and suggestions for future work.

This chapter summarizes the main contribution of this thesis, a grounded theory of Mob Programming for writing code with high complexity and remote work. This is followed by a discussion of the relationships between the roles, practices, and aspects involved. The next two sections critique our grounded theory and identify the limitations of this study. This is followed by a discussion of the theory in light of existing literature, implications for practice, and suggestions for future work.

9.1 Research Contributions

This thesis presents a grounded theory of Mob Programming for writing code with high complexity and remote work. This theory is based on a Grounded Theory research study involving 200 programmers from different software organizations in Brazil and USA over a period of seven years. The theory of Mob Programming to Complexity explains how software development teams take on informal, implicit, transient, and spontaneous roles, and perform balanced practices while facing critical environmental factors, in order to become an effective team delivering value write code with high complexity and remote work.

The long-term effects of satisfaction and an understanding of the satisfaction of the programmers are vitally significant to top management. Because it could help in the reduction of turnover, as well as maintain the professionals talented in the organization.

Ontologies provide agents with the qualification to start communication in a well effective way and with unprecedented immediacy. Through an agreed upon ontology, a group of agents can build a tailor-made Communication Space or accept an off-the-shelf one and immediately start coordinated action or collaborative problem solving da Silva and Agusti-Cullell (2008). We proposed the following Ontologies:

- Mob Programming Team Roles Ontology;
- Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors;

Perhaps, the main contribution was starting to write the Mob Programming Team Roles Ontology because writing the Team Roles as an Ontology helps those who are beginning to use Mob Programming guiding them and who have more experience with Mob Programming could aid by the addition of new roles or contribute to this field of study more scientifically.

Writing an Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors was a great challenge. However, I hope to help software developers from every country could have a better, safer, and more innovative Communication Space to work.

List the Challenges to Start Adopting Mob Programming in the chapter 6 "Challenges to Start Adopting Mob Programming" is also a contribution for new adopters of Mob Programming to learn from the mistakes of the Mob Programming pioneers.

Findings Related to Managing the Complexity categorized in the chapter 7 "Findings Related to Managing the Complexity" help to understand details about Mob Programming to be helpful in complex tasks.

Results About Remote Work categorized in the chapter 8 "Results About Remote Work" help to understand details about Mob Programming to be helpful in remote work.

9.2 Mob Programming Team Roles

Well-Defined Team Roles are the Programmers when the team decides to adopt the Mob Programming Team Roles Ontology, choosing the appropriate Roles for their Context. It observed the positive effect of having Well-Defined Team Roles is helps all team members keep the focus on problem-solving.

The theory explains how software development teams take on informal, implicit, transient, and spontaneous roles; perform balancing acts on a set of integrated practices to become an effective Mob Programming team. The Roles are Driver, Navigator, Researcher, Facilitator, Nose, Quality Analyst, and CyberSecurity Analyst.

By explicitly acknowledging this Ontology on Mob Programming Team Roles, the entire team could play to people's strengths and celebrate how their individuality can best contribute to the overall effectiveness and flow of the Mob Programming Session.

The Grounded Theory of Mob Programming Team Roles, beyond the fact of the theory, explains how software development teams take on informal, implicit, transient, and spontaneous roles; perform balancing acts on a set of integrated practices to become an effective Mob Programming team. It was fundamental for understanding how and why Mob Programming works so well in complex tasks and remote work.

The roles described in the following subsections are Driver, Navigator, Researcher, Facilitator, Nose, Quality Analyst, and CyberSecurity Analyst; In the following subsections, we describe the informal roles which exist in Mob Programming teams. The Roles in Mob Programming teams are informal and implicit. Differently from the organizational Roles, Mob Programming Team Roles are not designated to the individuals who perform them. Therefore, no problem related to the rotation of these roles.

9.3 Mob Programming Practices, Mob Programmers' Behaviors, and Mob Programming Communication Space

The theory explains how software development teams perform balancing acts on a set of integrated practices, in order to become an effective Mob Programming team. The integrated practices are Driver Rotation, Retrospectives, Structure breaks, Group Size, Room setup, and Automation. By explicitly acknowledging this Ontology, the whole team could suggest people's behaviors to better contribute to the overall effectiveness and flow of the Mob Programming Session and the Communication Space helping the Mob Programmers to have a Productive Dialogue.

We summarized the notions of Communication Space and Productive Dialogue. The reason is they are helpful in the understanding of the practices and Mob Programmer's behaviors. A Communication Space is an agreed-upon configuration of terms to be shared by a group of agents, through which they interact in such a way that the consequences of the interaction are (at least partially) predictable for all members of the group. A Productive Dialogue is such that the participation

agents can coordinate their actions. The agents participating in a productive dialogue must not necessarily act together to reach a common goal.

Driver rotation techniques, as Facilitators for productive dialogues. Through an Initial Information System, Timer, Standardized Procedures, Strong-Style, Red/Green Criteria, and a Facilitator.

Team Members Rotation among the distinct Mob Programming sessions. If a team member is not contributing, it should be moved to another Mob Programming session.

Retrospectives helps to improve the Mob Programming session.

> "|*One of the changes resulting from a mini-retrospective was the revision of the piloting time, which were very useful for adjusting the route. We used a pre-defined rotation time controlled by the facilitator. During the retrospective sessions, we had an analysis in the time [for Drive Rotation], initially 10 minutes for pilotages, and it was increased to 15 minutes after the first retrospective.*|" P31

Remove one person, and the team maintains a good performance, except in some cases, e.g., if the person absent it was senior and more than one.

Structure Breaks are fundamental and fruitful to managing complexity.

Behavioral positive response to the satisfaction of programmers Contributing to managing the complexity and remote work.

The courage of programmers is a behavioral positive response to Contributing to managing the complexity and remote work. Kindesse help to create a safe Communication Space, allowing the team members showing theirs vulnerabilities.

This is crucial because always the programmers need to learn something new. And, expose what the person do not knows requires vulnerability. When a programmer do not know the programming language being used at a moment when this person is the Driver, a fearlessness attitude helps because all team members contributing this programmer learn quicker. Contributing to creation of a Safer, Innovative Communication Space wherer people Embracing the Vulnerabilities with Kindness.

9.4 Challenges to Start Adopting Mob Programming

These categories are grounded in the data observed, questionnaires applied, and interviews done. Advocates of Mob Programming, including the author of this research, often struggle to convince their senior managers or other programmers to make Mob Programming part of a daily work routine in the team. My Master of Science Dissertation is about a Pair Programming extension, permitting it simultaneously and further pursuing the goal of improving the economic aspects implicated in the practice. And, I did hear in the last twenty-seven years from the Senior Management about paying twice for the same code. The same old common question of senior management that comes up frequently in this situation concerning the economic aspects of the practice: Does Mob Programming just cost much more money, and is it ultimately worth the extra cost because of the increased quality and team benefits? There are no studies on the topic. However, this study of Mob Programming could be cited for providing evidence that it is worthwhile to use it. Software development is a process full of change and uncertainty, with many aspects beyond lines of code, and many factors related to productivity in software development that are hard to compare and measure, e.g., analysis, testing, or quality.

When the team use mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. It was observed a correlation with the leading challenge of keeping the whole Mob Programmers focused on solving the problem.

When people with different experience levels are together in the same Mob Programming session, this often leads to false assumptions on how much each of them can contribute, or frustrations because of difference in pace. It is more challenger to get a Productive Dialog.

> "*Yes, it is a lot more exhausting than regular work. We took 15-20 minutes breaks together, and we also had occasions where a person would step back and handle something to, later on, return to the mob.*" P15

Working so closely with another person for long stretches of time is intense. You need to communicate constantly and it requires empathy and interpersonal skills.

9.5 Roles-Practices-Behaviors-Communication Space relationships related to Complexity

Without a deep understanding of the information flow and appropriate knowledge sharing, the complexity will grow up lacking the correct webs of sense-making. Therefore, it could be crippling. Productive Dialogues about Source-Code Quality and Deliver Value helps to manage the complexity.

Knoledge Sharing helps to manage the complexity because having more people work on a piece of the code helps the team spread knowledge on technology and domain on a daily basis and prevents silos of knowledge. Moreover, when two minds understand and discuss a problem, we improve the chances of finding a good solution. Different experiences and perspectives will lead to the consideration of more alternatives.

When using Mob Programming, the team has more eyes on the simple and complex details of the code written. Thus, more errors will get caught on the way instead of after the code is typed.

Mob Programming limits the number of things a team can work on in parallel, and therefore increases the overall focus. This will ensure that work constantly flows, and that blockers are addressed immediately.

Mob Programmers makes sure that every line of code was touched or seen by at least the major part of the people in the Mob Programming session. Therefor, it helps to manage the complexity.

Behavioral positive response to the satisfaction of programmers Contributing to managing the complexity.

If you want quality, you need to build it. Our proposed theory build quality into the Software because inspections are too late for quality. Software quality is a necessity to be in the development of the software product. A quality assurance analyst's job usually is inspection. Thus, it is too late. Mob Programming builds quality into the product instead of doing a verification.

9.6 Limitations

This subsection describes the limitations of this study, which are mainly the Grounded Theory limitations and research techniques used in the Data Collection. Additionally, we identified limitations regarding the number of participants, ontologies, and related to be the first study of this kind in the field.

According to Adolph *et al.* (2008) the inherent **limitation** of a GT study is that the resulting theory can only be said to explain the specific contexts explored in the study. Since the codes, concepts, and category emerged directly from the data, which in turn was collected directly from real world, the results are grounded in the context of the data.

Data derived from interviews is prone to **bias** according to Parry (1998). We did build the ability and skill to be able to ask questions that can counter-check the data provided. We also ensured the authenticity of the data collected through interviews by supplementing it with observations of workplaces and activities. The data derived from observations were not contradicted but rather supported the interview data, thereby strengthening it. Conducting interviews with open-ended questions allowed the researcher to ask the participants for specific and detailed examples. Also, it helped to ask a question in multiple ways at different points in the interview.

As with any empirical software engineering, the very high number of variables that affect a real software engineering project make it difficult to identify the impact that any one factor has

on the success or failure of the project. The Weel-Definied Team Roles; Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists; Kind and Fearlessness Programmer, and Structured Breaks, however, were clearly evident.

9.6.1 Number of Participants

A limitation of this research study is that the contexts studied were dictated by the acceptance of research-participating organizations, institutions, and companies, which in time, were in some ways limited by our access to them. Similarly, the selection of research participants was restricted by their willingness to participate. Moreover, the participants must believe in the potential benefits of Mob Programming or at least give it a try.

9.6.2 Ontologies

One evident problem related to ontologies is the strong requirement they implicitly demand, that agents accept the rules of interaction proposed in that third-party product in order to interact. Virtual environments that enable groups of agents to collectively and dynamically build the ontology they intend to use for their interactions, thus adding to the ontologies novel degree of flexibility purpose-driven interaction among groups of agents Gruber (2007).

9.6.3 Pioneer

The grounded theory of Mob Programming generated in this research is a first of its kind in the field. Further research into Mob Programming teams in Agile software development and other disciplines will help generate a more generalized theory.

9.7 Discussion

This section discusses our results in light of related literature to help us in the quest for correlations between our theories with other existing ones.

9.7.1 Team Roles

When the team uses mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Was observed a correlation with the challenge of keeping the whole Mob Programmers focused on solving the problem. The positive effect of having "Well-Defined Team Roles" is contributing to all team members keeping the focus on problem-solving. When the team uses only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. We observed a correlation in the leading challenge of keeping the whole Mob Programmers focused on solving the problem.

Driver or **Pilot** is the person typing on the keyboard, having the input devices, and primarily working on implementing the next step agreed upon with the Navigator or Co-pilot and does not spend time thinking because it could stop the whole team. The knowledge and skills in the programming language of the driver/pilot also are connected with the way the navigator guides the driver/pilot because the navigator/co-pilot depends on the abilities of the driver/pilot.

The **Navigator** or **Co-Pilot**, who is the lightning arrester for the Mob Programmers' ideas, primarily works on helping the next step agreed upon with the Driver/Pilot and protect the Driver and help the Driver not spend time thinking because it could stop the whole team. Stop the entire Mob is damager than only one person in solo programming. Therefore, the Navigator/Co-Pilot's principal responsibility is to guarantee the strategy of the next implementation helping the Driver in the progress of the codebase and being the lightning arrester for the Mob Programmers' ideas.

The **Facilitator** or **Mentor** performs a role in Remote Work regarding the Productive Dialogue in the online chat. The facilitator helps the team set time limits for each task. Facilitator Keep the team focused on the goal and contributes to removing any obstacles and addressing any

team dynamics. Also, the Facilitator protects the team from any outside interruptions and distractions.Therefore, the Facilitator responsibility is, e.g.,asking someone who is noticed a minor syntax improvement to hold that thought and wait until the tests are passing. Guarantee the Navigator being the lightning arrester for the Mob Programmers' ideas. While the driver and navigator focus on solving the issues of the programming, the Facilitator observes and manages the group's Communication Space and Productive Dialogue. When the tests are passed, the Facilitator is responsible for increasing the team morale. The Facilitator does not need to be a stand-up comedian but only remembers to ask everyone: "Are we happy with the code as it is? Can anyone suggest a refactoring at this point?". The objective of doing this is to contribute to a safer and more innovative Communication Space. This is also the person who remembers to hold the team to any rules or agreements they have made. For example, asking someone who is noticed a minor syntax improvement to hold that thought and wait until the tests are passing. They can also monitor the group for kindness and respect. Sometimes the problems we face as a mob, especially when diagnosing a problem nobody understands, can cause anxiety. Under these circumstances, the facilitator role is to notice this and guide the group back to a kind and respectful dynamic.

The **Researcher** or **Scout** role can be done by someone on the team who has used the law of two feet to make the best use of their time. Or, they check out of the mob for a time and explicitly go and focus on that task. Or, they research in solo time to find the next best thing customers are waiting for. With innovative and fresh ideas, they make future success possible. They go off and gather information: they read documentation, they write spikes, they talk to customers, they draw wireframes. Researches ideal profiles are those who venture into uncharted territories and come up with exciting ideas that haven't yet been done. They are bold, brave starters who have no fear of failing. The Researcher is curious by nature. In other words, this role fits better for someone who loves to explore. They look at the territory ahead and anticipate challenges and puzzles the team probably faces soon.

Quality Analyst or **Quality Guardian** brought functionality validation scenarios, in addition to reinforcement of the need to build unit tests. Work on the Code Quality Maps.

Nose or **Housekeeper** works on a Map which permits not stopping the whole group when finding some code smell or quality problem. At an appropriate moment, they could fix it in the future using the Map.

CyberSecurity Analyst or **Hacker** work on Security Lists and Hardening checklists. Human Mob Programming is working on security issues instead of an Automated CyberSecurity Risk Management tool.

Well-Defined Team Roles are the Programmers when the team decides to adopt the Mob Programming Team Roles Ontology choosing the appropriate Roles for their Context. It observed the positive effect of having Well-Defined Team Roles is helps all team members keep the focus on problem-solving.

9.7.2 Team Practices and Team Members' Behaviours

Anderson and McMillan (2003) examine fundamental issues of teamwork, and question whether such teams really are analogous. After detailing some of the striking similarities, they conclude that they are indeed equivalent and comparable. Thus, their work is a preliminary study into whether nature-specifically, insect societies-may provide not just a valid metaphor but, moreover, a model for organizational shift and effective function in human enterprise.Thus, a possible metaphor about the strategy of only sometimes work with the entire team, are the bees that do not swarming all the time, but only in specifics moments: lack of space for posture or to store the honey and pollen, etc. Other possible metaphor is collective sports, one example of sport with ball is the soccer, although the focus of the game is around the ball, in a good team always has players playing without the ball, following tactics functions. Therefore, these tactics functions are also fundamental for Mob Programming teams.

Western Eletric made experiments attempted to determine the relationship between light levels and worker efficiency. The data compiled by the Illumination Experiments indicated only a minor

correlation between light levels and worker productivity. However, one interesting observation is sometimes when increase the light level, the productivity grow up and when decrease the light level the procuctivity grow up too Mayo (1933).

A possible explanation is an external factor that in some way improve the bond formed among the members maybe could be positive to the productivity of the team. But, metaphorsly by the last time, medicine is poison depending on the dose.

> "This helps explain the fact observed when occurring Mental Exhaustion is a good idea to alternate to a structured break, mini-retrospective, pair programming, or simultaneous style pair programming."

9.7.3 Evaluating a Grounded Theory

Although I found Grounded Theory a fittable research method, it helped me find the answers I was looking for as to why Mob Programming works so well for complex tasks and whether it could be used remotely. I like to record just one and the following observation, leaving to write the thesis text only at the end in case of a doctoral student. It can be complicated due to the deadline. Perhaps after some categories are stable, I would advise Ph.D. students who have chosen to use Grounded Theory about as soon as some core categories are firm that they start writing their text. By the way, I went through a pandemic doing field research. I had to ask for an extension three times.

9.8 Implications for Practice

Our theories have several implications for practitioners. The following sections present the evident impact of our contributions on the teams, and their senior management to designing a safer and more innovative Communication Space.

9.8.1 Implications for Teams

The transition to becoming an effective Mob Programming team is not easy. The roles and practices described in this thesis should help team members understand their roles and behaviors when becoming a productive Mob Programming team.

One of the characteristics of Mob Programming teams is their ability to react positively in response to challenges. In an innovative Communication Space, teams can expect to get involved in a safer environment to evolve themselves and give their best contribution possible to the team. These practices include Retrospectives, Structured Breaks, Well-Defined Team Roles, and so on, to promote transparency, collective decision-making, and self-assignment.

The popularity of Mob Programming has led to several project managers from traditional software development backgrounds giving it a try. Creating an Innovative Communication Space is not easy, and an agile coach could help with this endeavor. The acting through the Team Role of Facilitator collaborates with the Team Members and acts fearless when exposing some vulnerability or contributing to the Team Members being Kind to each other. Perhaps, at the right time, suggest a Structure Break or alternate with other practices and back and forth among the approaches.

Facilitators could help in guiding their team into a safer and more innovative Communication Space. The critical factors identified in this thesis should help Facilitators know what to expect in terms of challenges and how to react through the Mob Programming Team Roles, Practices, and Mob Programmer's behaviors. An important implication for the Facilitator, however, is to always be mindful of the adoption of the Ontology of the Mob Programming Team Roles of these and the Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors. Thus, a facilitator or mentor can facilitate the emergence of new roles and practices rather than forcing them on the team.

9.8.2 Implications for Senior Management

Senior management must be aware of their influence on the ability of Mob Programming productivity in complex tasks and remote work. An aspect of this awareness is understanding Mob Programming and their role as Senior Managers in creating an innovative Communication Space. Moreover, they may subsidize a conducive environment for Mob Programming teams to achieve and sustain their potential. The organizational culture should be expressed transparently, with trust, openness, free Information Flow, and Knowledge Sharing. Senior management must decide whether such changes are beneficial for their organization.

9.9 Implications for Academia

A pattern is the formalization of a problem/solution pair used to make an object-oriented design decision. The purpose of a pattern is to codify existing design knowledge so that developers are not constantly reinventing the wheel. Also, naming these patterns eases communication among developers. The pattern definition is a solution to a problem in a context. The different pattern forms all have particular qualities to them, and any patterns author will tend to pick a form that works well with their innate preferences. Despite the various forms, most patterns do have common elements. An instantiated ontology is a communication space used for collaborative problem solving da Silva and Agusti-Cullell (2008). Ontologies are possible alternatives to Patterns. I truly believe that Ontologies are more appropriate ways to write in detail the nuances that Collaboration in Software Development requires.

> "I hope my colleagues agree with me and allow **Ontologies** because I would appreciate having more **Ontologies** to read instead of only **Patterns**."

Ontologies provide agents with the qualification to start communication in a well effective way and with unprecedented immediacy. Through an agreed upon ontology, a group of agents can build a tailor-made Communication Space or accept an off-the-shelf one and immediately start coordinated action or collaborative problem solving da Silva and Agusti-Cullell (2008). The two proposed Ontologies are following briefly described in Table 9.1:

Ontology	Description
Mob Programming Team Roles Ontology	Perhaps, the main contribution was starting to write the Mob Programming Team Roles Ontology because writing the Team Roles as an Ontology helps those who are beginning to use Mob Programming guiding them and who have more experience with Mob Programming could aid by the addition of new roles or contribute to this field of study more scientifically.
Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors	Writing an Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors was a great challenge. However, I hope to help software developers from every country could have a better, safer, and more innovative Communication Space to work.

Table 9.1: *This table contains only a short description of the two Ontologies since more details beyond this brief description of Mob Programming Team Roles Ontology are available in Chapter 4, and about the Ontology of Mob Programming Communication Space, Mob Programming Practices, and Mob Programmers' Behaviors in Chapter 5.*

Following are the first four Theories, with their brief description in Table 9.2. Each short description has a link to the respective section that contains the full description of the Theory using 6C's model. The number of discovered theories is six, and due fit the page, it was split into two tables to make it easy to read.

Theory	Description
Well-Defined Team Roles Theory	When the team uses mainly only the roles of Driver/Pilot and Navigator/Co-Pilot well defined. Was observed a correlation with the challenge of keeping the whole Mob Programmers focused on solving the problem.
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable	Structure Breaks are fundamental and fruitful in managing complexity and preventing Mental Exhaustion. Five different contingencies were identified: Learning something new together in a group or solo; Mini-Retrospectives; Pair Programming; Members Rotation among the Teams; Pair Programming Simultaneous Style.
Embracing Vulnerabilities with Kindness Theory	We identified the positive effect of having a Kind and Fearlessness Programmer because this behavior contributes toward a Safe and Innovative Communication Space.
Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming	We identified the positive effect of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists that contribute to outcomes or goods of having Code Quality and Code Smells Maps; Security Lists and Maps; Hardening Checklists, these benefits are: closer team; knowledge sharing; manage complexity; higher quality software; adherent to the code standard; clean code and safer; reducing the chance of a damaging cyber incident; maximizing resilience.

Table 9.2: *This table contains only a short description of the first four Theories since more details beyond this brief description of Well-Defined Team Roles Theory are available in Section 4.10, and about the Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable in Section 5.17, and about Embracing Vulnerabilities with Kindness Theory, in Section 6.10, and about the Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming in Section 7.15.*

Following are the last two Theories, with their brief description in Table 9.3. The total number of discovered theories is six, and due fit the page, it was split into two tables. The last two Theories are related to Remote Mob Programming. **Perhaps, an implication for academia is these six Theories were discovered from the Classical Version of Grounded Theory outside of Social Science in the Computer Science field.**

Theory	Description
A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage	Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors
Theory of Benefits of Remote Mob Programming	Outcomes or effects of having Remote Mob Programming are that the Mob Programming benefits are preserved; the possibility of working from anywhere; family and pets closer; little-to-no-commute; greater inclusivity; non-stop working in different time zones.

Table 9.3: *This table contains only a short description of the last two Theories since more details beyond this brief description about a theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage are available in Section 8.12, and about the Theory of Benefits of Remote Mob Programming are available in Section 8.13.*

Using the 6C's model we developed 6 Theories related to our Grounded Theory about Mob Programming for writing code with high complexity and remote work. The Ontologies were fundamentals to rationalize the theories about Complex Problem Solving, Remote Work, and to Embracing Vulnerabilities with Kindness to making a safe and innovative Communication space. The six theories are:

- Well-Defined Team Roles Theory;
- Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable;
- Embracing Vulnerabilities with Kindness Theory;
- Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming;
- A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors;
- Theory of Benefits of Remote Mob Programming.

Collaboration and problem-solving are old challenges that humanity confronts. E. g., according to Britannica, the First Letter of Paul to the Corinthians, probably written about 53-54 CE at Ephesus, Asia Minor, deals with problems that arose in the early years after Paul's initial missionary visit (c. 50-51) to Corinth. Mob Programming is for problem-solving, and the power of Love was fundamental thousands of years ago and continues to be the most important.

da Silva and Agusti-Cullell (2008) it was a guiding star in the form of a book, leading me through the ocean of knowledge, and da Silva and Agusti-Cullell (2008) finished the book with the first letter of Saint Paul to Corinthians. Instead of a summary of complex ideas that hardly could be summarized without a fundamental lack of understanding of the theory. To honor the most innovative book I read in my whole life, firstly, I say thank you to the authors for writing and concluding this inspirational way. Therefore, I also finalize my Theories by citing the first letter of Saint Paul to Corinthians (see Appendix I.10). Love helps to deal with problems that arise to manage a Mob Programming session when solving complex issues. Love is the perfect behavior when a Mob Programmer shows vulnerability. Love is also courage when you are vulnerable. Communication Space fulfilled of Love is safe and innovative. Love understands everything, and it adapts itself. Love helps to manage the complexity and allows work from everywhere.

9.10 Future Work

- Open Source Voice Bot tool to be the Driver, in the beginning, at least in a specific programming language;
- Open Source Software to unify the Grounded Theory raw data, e.g., interview scrips, answers of the questionnaires. Beyond to centralize the all the data. Permiting Artificial Inteligence for suggest categories and theories. Automation of the anonymoius the data;
- Change Brazilian Law toward permitting public datasets, with the data anonymous, specifically in the Computer Science field. Exploration of the benefits of these public datasets to Computer science and the difference from Medicine field. Probably our datasets would be less sensitive about damage to individual privacy;
- Development of a WebSite for the Mob Programming Team Roles Ontology, permmitting the update of the Team Roles using the proposed writing pattern of this Ontology;
- All aspects researched about Pair Programming, e.g., gender, cost, effort, and so on, research about it using Mob Programming;
- CyberSecurity-related aspects when doing Mob Programming remotely.

Appendix A

Questionnaire in Portuguese

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

Questionário

Título da Pesquisa : Mob Programming: Medidas, Efeitos e Resultados

Pesquisador Responsável: Herez Moise Kattan

Nome do Questionário : Questionário sobre experiência com programação e Mob Programming

Nome do Arquivo : Questionario.pdf

As questões seguintes são pertinentes à experiência dos membros da equipe com programação, processos de desenvolvimento de software e tecnologias utilizadas pelo time usando o Mob Programming:

1. Quantos anos você tem de experiência com programação de computadores? (Experiência educacional e profissional em qualquer linguagem de programação).
2. Como você avalia seu nível atual de conhecimento na linguagem de programação usada pelo seu time? Quais linguagens de programação e tecnologias vocês usaram para produzir o software usando o Mob Programming?
3. Piloto, que esta na posse do teclado. Navegador, para-raios das ideias da equipe e comunica-se com o Piloto. Guardião da Qualidade, Inspetor de Segurança, Pesquisador, Facilitador. Descreva os papéis dos membros da equipe que usaram. Quais você achou importante? Por quê?

As próximas perguntas são pertinentes à configuração da sala e também solicitam uma descrição do ambiente no qual a sua equipe programou usando o Mob Programming. Se o ambiente onde sua equipe usou o Mob Programming pode ser aprimorado, por favor detalhe como melhorá-lo:

1. Vocês usaram uma grande televisão ou projetor? Quantos? Qual era o tamanho? Na sua opinião foi ideal ou poderia melhorar? Se sim, como poderia ser melhor a visualização?
2. Às vezes, outros computadores foram usados? Computadores adicionais são úteis?
3. Quantas pessoas havia na sala de trabalho? Todas as pessoas presentes eram do seu time?
4. Pessoas de outras equipes pediram ou ajudaram em algo? O barulho na sala foi um problema?
5. A visualização pode ser melhorada? A visualização do código foi adequada?
6. Vocês usaram um laser pointer ou outras formas para se comunicarem com o piloto?
7. Você poderia nos enviar fotos ou vídeos usando o Mob Programming? Se sim, todos os membros da equipe autorizam? Se sim, todos os membros têm que assinar o termo autorizando relacionado ao direito de imagem e voz.

Figure A.1: *First Page of the Questionnaire in Portuguese*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO

IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170

https://www.ime.usp.br

Estas questões são relacionadas à descobrir se uma pessoa precisa se ausentar na sessão de Mob Programming, o grupo mantém o mesmo desempenho:

1. Sobre a frequência de trabalho das pessoas, você acha que o Mob Programming melhora o engajamento? Satisfação dos membros da equipe? Confiança entre os membros da equipe? Ajuda a criar um bom ambiente de trabalho? Pode comentar algo sobre isso?
2. Ao usar o Mob Programming em algum momento alguém da sua equipe precisou se ausentar? Se sim, como ficou o desempenho da equipe com uma pessoa ausente?
3. Se a resposta for sim para a pergunta anterior, em algum momento mais de uma pessoa precisou se ausentar? Se sim, como ficou o desempenho da equipe com mais de uma pessoa ausente?

Estas questões são pertinentes à rotação da posse do teclado:

1. Vocês usaram um cronômetro ou um programa para rotação da posse do teclado? Como fizeram a rotação? Tem alguma sugestão para melhorar a rotação da posse do teclado?
2. Vocês usaram o estilo de rotação da posse do teclado "Strong Style", onde o piloto apenas digita e os co-pilotos dizem o que fazer e se o piloto quer dar uma ideia tem que entregar a posse do teclado para outra pessoa. Em outras palavras, o piloto não tem ideias e para que uma ideia passe da sua cabeça ao computador, ela deve passar pelas mãos de outra pessoa. Vocês usaram ou não o "Strong Style"? Nesse sentido, por favor, descreva a experiência da sua equipe ao rotacionar a posse do teclado e compare com outros estilos que sua equipe usou ou com os quais já teve experiência anterior, detalhando o funcionamento e a sua opinião a respeito.

Estas questões são pertinentes às retrospectivas e mini-retrospectivas:

1. A equipe fez mini-retrospectivas diárias após usar o Mob Programming? Foram úteis?
2. A equipe fez outras retrospectivas em um intervalo de tempo maior como semanalmente, quinzenalmente, mensalmente, bimestralmente, semestralmente ou anualmente? Foram úteis?
3. A equipe mudou alguma coisa depois de uma retrospectiva ou mini-retrospectiva (Sessão de Mob Programming)? O que mudou? As alterações foram úteis? Retrospectivas foram úteis? Qual a frequência ideal delas?

Em relação à automação do trabalho para evitar o tempo ocioso, pois é diferente desperdiçar um minuto do tempo de um membro da equipe e perder um minuto de toda a equipe:

1. A sua equipe automatizou alguma coisa? O que? Foi útil? Por quê?

Ao lidar com sistemas grandes e complexos, em determinadas situações é interessante focar em um determinado problema que está tentando resolver no momento, então abordar apenas o problema em questão e ignorar todas as outras preocupações pode ser uma boa ideia e ajudar

Figure A.2: *Second Page of the Questionnaire in Portuguese*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

em sua resolução. Objetivando reduzir o risco de um problema técnico no sistema grande e complexo ou aumentar a sua confiabilidade:

1. Em algum momento vocês se separaram para descobrir respostas para difíceis problemas técnicos de programação, interface ou projeto? Se separaram para criar um programa mais simples para explorar potenciais soluções para um problema complexo e depois se juntaram ao grupo novamente? Enfrentaram essa situação? Se sim, se separaram momentaneamente e depois retornaram ao grupo? Os membros de sua equipe se separaram e fizeram alguma solução menor para integrar de volta à solução principal?

Estas questões são sobre intervalos e se o envolvimento total o tempo todo pode ser exaustivo, principalmente em tarefas complexas:

1. O envolvimento total o tempo todo é exaustivo? Vocês fizeram intervalos juntos? Como foram?
2. A sua equipe se certificou de que a cada hora houvesse um intervalo de 5 a 10 minutos em que as pessoas não pudessem ficar atrás da tela? O que você acha disso?
3. Intervalos ajudaram na resolução de algum problema complexo? Poderiam ter ajudado?

Questão sobre aprendizado dos membros da equipe:

1. A sua equipe tentou aprender algo? Estavam juntos ou separados? Foi no início da sessão de Mob Programming? Como foi a experiência de aprendizagem? Você considera que o Mob Programming contribuiu para o seu aprendizado? O que você aprendeu usando o Mob Programming?

Estas questões são pertinentes à Inteligência Coletiva no desenvolvimento de software complexo:

Todo programador já passou por momentos de exaustão mental em que pode levar muito tempo para resolver até mesmo o problema mais trivial. A maioria dos programadores (mesmo aqueles que não praticam Pair Programming ou Mob Programming) está ciente disso e convida um colega programador(a) para dar uma olhada em seu código nesses momentos de exaustão ou irá procurar conversar, por exemplo, durante um intervalo tomando café com outro programador(a). Em uma sessão de Mob Programming, esses momentos são praticamente inexistentes.

1. A sua equipe concorda com a afirmação acima? Foi unânime? Mob Programming ajudou a evitar pelo menos um pouco esses momentos de exaustão mental?
2. Aconteceram momentos de exaustão mental na sua equipe? Como vocês lidaram? O que você achou da forma como vocês lidaram?
3. Houve algum membro da equipe que sentiu que o Mob Programming estava atrasando o trabalho ou era um desperdício de tempo?

Figure A.3: *Third Page of the Questionnaire in Portuguese*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP
Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

Ao lidar com questões de maior alcance, como escalabilidade e arquitetura, o Mob Programming provoca um diálogo criativo com uma troca de ideias sem discussões longas ou acaloradas. Depois de um tempo, nos casos em que uma diferença significativa de pontos de vista persiste por mais tempo do que o normal, as decisões coletivas são baseadas na confiança mútua.

1. A equipe concorda com a afirmação acima? Por favor, diga-me algo sobre isso com base na experiência da sua equipe.
2. A sua equipe foi trabalhando melhor junto com o passar do tempo? Isso gerou um aumento no aprendizado individual e coletivo? Gerou satisfação e aprendizado?

Estas questões são pertinentes à Aprendizagem e compartilhando de conhecimento técnico e do domínio do problema:

No desenvolvimento de sistemas inovadores e complexos, a aprendizagem deve ser parte integrante da atividade diária de trabalho. O Mob Programming fornece um ambiente ideal para isso. Ao observar os outros, os membros da equipe aprendem quase que imediatamente como fazer melhor uso das ferramentas e realizar seu trabalho diário. Mais importante, eles logo obterão uma compreensão mais profunda das regras de negócio e do domínio do problema e consequentemente melhoraram suas habilidades de resolução de problemas à medida que observam e questionam outras pessoas sobre as soluções propostas.

1. Você concorda com a afirmação acima?
2. Por favor, descreva se o Mob Programming ajudou na compreensão do domínio do problema e melhorou a competência técnica do time com base na experiência de sua equipe.

Outras práticas foram usadas em conjunto com o Mob Programming:

1. A sua equipe usou o Mob Programming o tempo todo? Houve atividades em que a sua equipe não aplicou o Mob Programming? Por quê?
2. Usaram outras práticas em conjunto com o Mob Programming? Por favor, conte-nos sobre seu espaço de trabalho informativo ou sobre outras práticas que vocês adotaram.
3. A equipe acha que é bom programar usando o Mob Programming o tempo todo?
4. A sua equipe mediu algo? O que e como foi medido?

Estas questões são pertinentes à Complexidade:

1. Você acha que o Mob Programming foi benéfico em tarefas complexas? Por quê?
2. Melhorou a qualidade do software? A sua equipe mediu? Como foi medido?

As seguintes questões são pertinentes à Satisfação:

1. Você ficou satisfeito com o Mob Programming? Prefere trabalhar sozinho, em pares ou em grupo?

Figure A.4: *Fourth Page of the Questionnaire in Portuguese*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

2. Por favor, comente sobre os pontos positivos e negativos do Mob Programming na sua experiência.
3. Quais foram as experiências favoritas da equipe relacionadas com o Mob Programming?
4. Os membros da sua equipe estão satisfeitos e aprovaram o Mob Programming? Foi unânime?

As próximas questões estão relacionadas ao trabalho remoto:

1. Durante a pandemia, onde todos foram obrigados a trabalhar remotamente, o Mob Programming foi benéfico? Pode destacar alguns aspectos relacionados com esse momento tão crítico?
2. Ao usar o Mob Programming remotamente, quais softwares vocês usaram?
3. O número ideal na sua opinião de pessoas trabalhando em grupo usando o Mob Programming presencialmente é o mesmo que remotamente? Quantos? Por quê?

Estas últimas questões são pertinentes à Melhoria Contínua e trabalho na Pandemia:

1. Quantas pessoas você considera ideal para programar usando o Mob Programming?
2. Você acha que usar o Mob Programming foi produtivo? Foi mais produtivo do que outras abordagens que você já usou para trabalhar com um grupo com o mesmo número de pessoas?
3. Algum aspecto do Mob Programming a sua equipe conseguiu melhorar? Pode ser técnico, software, comportamental da equipe, padrão de colaboração da equipe, do piloto, ou co-pilotos, definição de papéis (piloto, co-piloto, pesquisador, tomador de notas, do contra, facilitador, etc).
4. A configuração da sala antes da pandemia como era? Pode ser melhorada? No pós-pandemia você se sente seguro em trabalhar em grupo presencialmente usando o Mob Programming? No pós-pandemia, quando retornarem ao trabalho presencial, vocês pretendem continuar a usar o Mob Programming presencialmente?
5. Ainda sobre o pós-pandemia, vocês vão adotar algumas medidas como divisória de acrílico separando os programadores, uso obrigatório de máscaras, álcool em gel, apresentação do comprovante de duas doses da vacina e testes semanais de COVID? Você gostaria que tais medidas fossem adotadas? Tem alguma sugestão de medida adicional que você gostaria que fosse adotada?
6. Quais foram as dificuldades ao começar a usar o Mob Programming? Quais foram as dificuldades ao começar a usar o Mob Programming remotamente?
7. Comente por favor, especificamente sobre as dificuldades e desafios para o trabalho que a pandemia causou. Além disso, em sua opinião, usar o Mob Programming ajudou a melhorar ou não a experiência de programação, considerando os aspectos sociais de não estar programando sozinho e sim conectado o dia inteiro com um grupo de programadores (as) usando o Mob Programming.

Figure A.5: *Fifth Page of the Questionnaire in Portuguese*

Appendix B

Questionnaire in English

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo SP-Brazil- Telephone: +55 11 3091-6170 | https://www.ime.usp.br

QUESTIONNAIRE

Research Title : Mob Programming: Measures, Effects, and Results
Researcher Name : Herez Moise Kattan
Questionnaire Name: Questionnaire about the experience with programming and Mob Programming
File Name : Questionnaire.pdf

The following questions pertain to team members' experience with programming, software development processes, and technologies used by the team using Mob Programming:

1. How many years of experience do you have with computer programming? (Educational and professional experience in any programming language).
2. How do you rate your current level of knowledge in the programming language used by your team? What programming languages and technologies did you use to produce the software using Mob Programming?
3. The pilot types on the keyboard. Navigator, lightning rod for the team's ideas and communicates with the Pilot. Analyst of Quality or CyberSecurity, Researcher, and Facilitator. Describe the roles of team members used. Which ones did you find important? Why?

The following questions are about the room setup. Also, these questions look for a description of the environment in which your team developed software through Mob Programming:

1. Did you use a large television or projector? How many? What was the size? In your opinion, could it improve? If so, how could the visualization be better?
2. Were other computers sometimes used? Are additional computers useful?
3. How many people were in the workroom? Were all the people present from your team?
4. Did people from other groups ask for help or help with something? Was noise in the room a problem?
5. Can the visualization be improved? Was the code preview adequate?
6. Did you use a laser pointer or other ways to help in the communication with the pilot?
7. Could you send us photos or videos using Mob Programming? If yes, do all team members authorize it? If yes, all members must sign the authorization term related to image and voice rights.

These questions are related to finding out if a person needs to be absent from the Mob Programming session, the group maintains the same performance:

1. About people's work frequency, do you think Mob Programming improves engagement? Team member satisfaction? Trust among team members? Does it help to create a good working environment? Can you comment something on this?
2. When using Mob Programming, did someone on your team need to leave at any time? If yes, how was the team's performance with one person absent?
3. In the previous question, if it answered yes. Did more than one person need to be absent at any time? If yes, how was the team's performance with more than one person away?

Page **1** of **5**

Figure B.1: *First Page of the Questionnaire in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo SP-Brazil- Telephone: +55 11 3091-6170 | https://www.ime.usp.br

These questions pertain to keyboard rotation:

1. How did you rotate the keyboard? Did you use a timer or a custom program? Do you have any suggestions to improve the rotation of the keyboard?
2. Did you use Strong Style? In this regard, please describe your team's experience with the rotating keyboard and compare it with other styles your team has used or has had previous experience with, detailing how it works and what you think about it.
   In the Strong Style keyboard rotation, if the pilot wants to give an idea, he needs to hand the keyboard to someone else. In other words, the pilot has no arguments, and for an idea to pass from a head to the computer, it must pass through someone else's hands.

These questions are pertinent to retrospectives and mini-retrospectives:

1. Did the team do daily mini-retrospectives after using Mob Programming? Were they helpful?
2. Did the team do other retrospectives over an extended period, such as weekly, biweekly, monthly, bi-monthly, semi-annually, or annually? Were they helpful?
3. Did the team change anything after a retrospective or mini-retrospective (Mob Programming Session)? What has changed? Were the changes helpful? Were retrospectives helpful? What is their ideal frequency?

Regarding the automation of work to avoid downtime, it is different to waste a minute of a team member's time and lose a minute of the entire team:

1. Has your team automated anything? What? Was it helpful? Why?

When dealing with large and complex systems, in certain situations, it is interesting to focus on a particular current problem, so just addressing the issue at hand and ignoring all other concerns can be a good idea, thus, helping in its resolution. To reduce the risk of a technical problem in a large and complex system or increase its reliability:

1. Have you ever parted ways to discover answers to complex technical programming, interface, or design problems? Did they split up to create an uncomplicated program to explore potential solutions to a complex issue after rejoining the group? Did you face this situation? If so, did they momentarily separate and then rejoin the group? Have team members split up to make smaller solutions to integrate back into the principal solution?

These questions are about breaks and whether full involvement all the time can be exhausting, especially on complex tasks:

1. Is full involvement at all times exhausting? Did you take breaks together? How went?
2. Did your team make a break every hour? there was a 5-10 minute where people could not be behind the screen. What do you think?
3. Did intervals help in solving a complex problem? Could they have helped?

Figure B.2: *Second Page of the Questionnaire in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo SP-Brazil- Telephone: +55 11 3091-6170 | https://www.ime.usp.br

Team member learning question:

1. Did your team try to learn something? Were they together or apart? Was it at the beginning of the Mob Programming session? How was the learning experience? Do you consider that Mob Programming contributed to your learning? What did you learn using Mob Programming?

These questions are pertinent to Collective Intelligence in complex software development:
Every programmer has gone through moments of mental exhaustion when it can take a long time to solve even the most trivial problem. Most programmers are familiar with this issue, including those who do not practice Pair Programming or Mob Programming. It is common to invite a fellow programmer to take a look at their code in these moments of exhaustion or will seek to talk, for example, during a break taking coffee with another programmer. In a Mob Programming session, these moments are practically non-existent.

1. Does your team agree with the above statement? Was it unanimous? Did Mob Programming help you avoid at least a little bit to handle these moments of mental exhaustion?
2. Have there been moments of mental exhaustion in your team? How did you cope? What did you think of the way you guys handled it?
3. Were there any team members who felt that Mob Programming was delaying work or was a waste of time?

When dealing with far-reaching issues such as scalability and architecture, Mob Programming sparks a creative dialogue with an exchange of ideas without long or heated discussions. After a while, in cases where a significant difference in views persists longer than customary, collective decisions are based on mutual trust.

1. Does the team agree with the above statement? Please tell me something about this based on your team's experience.
2. Has your team worked better together over time? Has this generated an increase in individual and collective learning? Did it increase satisfaction and apprenticeship?

These questions are pertinent to Learning and sharing technical and problem-domain knowledge:
In the progress of innovative and complex systems, learning must be an integral part of daily work activity. Mob Programming provides an ideal environment for this. By watching others, team members learn in real-time how to make better use of the tools and get their day-to-day work done. More importantly, they will soon gain a deeper understanding of the business rules and problem domain and consequently improve their problem-solving skills as they observe and question others about proposed solutions.

1. Do you agree with the above statement?
2. Please, describe if Mob Programming helped in understanding the problem domain. Based on your team's experience, if your team observed an improvement in technical competence.

Figure B.3: *Third Page of the Questionnaire in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO

IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo SP-Brazil- Telephone: +55 11 3091-6170 | https://www.ime.usp.br

Questions about if different techniques have been used together with Mob Programming:

1. Did your team use Mob Programming all the time? Were there activities where your team did not apply Mob Programming? Why?
2. Did you use other practices in conjunction with Mob Programming? Please, tell us about your informative workspace or other techniques you adopted.
3. Does the team think it's okay to program using Mob Programming all the time?
4. Did your team measure something? What and how was it done?

These questions are pertinent to Complexity:

1. Do you think Mob Programming was beneficial in complex tasks? Why?
2. Has the quality of the software improved? Did your team measure? How was it calculated?

The following questions are relevant to Satisfaction:

1. Are you satisfied with Mob Programming? Do you prefer to work alone, in pairs, or in a group?
2. Please, comment on the positives and negatives of Mob Programming in your experience.
3. What were the team's favorite experiences related to Mob Programming?
4. Are your team members satisfied and approved of Mob Programming? Was it unanimous?

The following questions are related to remote work:

1. During the pandemic, where people were due to work remotely. Was Mob Programming beneficial? Can you highlight some aspects related to this critical moment?
2. When using Mob Programming remotely, what software did you use?
3. In your opinion, is the ideal number of people working in groups using Mob Programming in person the same as remotely? How many? Why?

These last questions are pertinent to Continuous Improvement and work during the Pandemic times:

1. How many people do you think are ideal for programming using Mob Programming?
2. Do you think using Mob Programming was productive? Was it more productive rather other approaches you have used to work with a group of the same number of people?
3. Is there any aspect of Mob Programming that your team was able to improve? It can be technical, software, team behavior, team collaboration pattern, pilot, or co-pilot, and the definition of roles, e.g., pilot, co-pilot, researcher, note taker, contrarian, facilitator, etc.
4. What was the room configuration before the pandemic? Can it be improved? In the post-pandemic, do you feel safe working in a group in person using Mob Programming?

Figure B.4: *Fourth Page of the Questionnaire in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo SP-Brazil- Telephone: +55 11 3091-6170 | https://www.ime.usp.br

Yet about Post-pandemic, when you return to face-to-face work, do you intend to continue using Mob Programming face-to-face?

5. Still, about the post-pandemic. Will you adopt measures such as an acrylic divider separating programmers, mandatory use of masks, gel alcohol, presentation of proof of two/three/four doses of the vaccine, or weekly COVID tests? Would you like such measures to be adopted? Do you have any suggestions for additional actions you would like to see taken?
6. What were the difficulties when starting to use Mob Programming? And what were the challenges when starting to use Mob Programming remotely?
7. Please comment specifically on the difficulties and challenges to work that the pandemic has caused. Moreover, in your opinion, using Mob Programming has helped to improve the programming experience or not, considering the social aspects of not programming alone but connecting the day with a group of programmers using Mob Programming.

Figure B.5: *Fifth Page of the Questionnaire in English*

Appendix C

Interview Script in Portuguese

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

ROTEIRO DA ENTREVISTA

Título da Pesquisa : Mob Programming: Medidas, Efeitos e Resultados
Pesquisador Responsável: Herez Moise Kattan
Nome do Roteiro : Entrevista sobre Mob Programming em tarefas complexas e trabalho remoto
Nome do Arquivo : RoteiroEntrevista.pdf

Primeiramente, muito obrigado por sua cooperação em participar da pesquisa sobre o Mob Programming. A sua experiência prática é muito valiosa para nos auxiliar a aprofundar o conhecimento científico sobre essa nova técnica de programação em Grupo chamada Mob Programming e o foco do estudo não é apenas reduzir o custo de desenvolvimento de software, mas também melhorar a experiência de programação para o programador(a). Assim, a pesquisa é uma Teoria Fundamentada nos Dados (Grounded Theory) sobre essa técnica de programação chamada Mob Programming com o objetivo de verificar em projetos reais os seus benefícios no contexto de Desenvolvimento de Software. Especificamente em tarefas complexas e trabalho remoto, ambos sobre programação de computadores. Relembrando que você pode abandonar a pesquisa a qualquer momento sem nenhum constrangimento ou prejuízo.

1. Você considera que o Mob Programming foi benéfico em tarefas complexas? Por quê?
2. Melhorou ou não a qualidade do software? A sua equipe mediu a melhora ou piora na qualidade do produto de software produzido usando Mob Programming? Como foi medido?
3. Você ficou satisfeito com o Mob Programming ou prefere trabalhar mais isoladamente ou prefere programar em pares? Depende da complexidade da tarefa?
4. Você achou o Mob Programming mais útil em tarefas complexas em relação às tarefas simples ou de complexidade média? Você acredita que vale a pena trabalhar usando o Mob Programming sempre ou é melhor apenas em tarefas complexas?
5. Vocês mediram o desempenho da equipe ou a qualidade do software produzido ao desenvolver uma tarefa complexa versus uma tarefa simples ou de complexidade média usando o Mob Programming? Se sim, como vocês mediram e quais foram os resultados?
6. A sua equipe tentou aprender algo? Estavam juntos ou separados? Foi no início da sessão de Mob Programming? Como foi a experiência de aprendizagem? Você considera que o Mob Programming contribuiu para o seu aprendizado? O que você aprendeu usando o Mob Programming?
7. Você avalia ser uma boa opção alternar entre Mob Programming e Pair Programming? Em relação ao compartilhamento de conhecimento e aprendizado são iguais ou diferentes? Gera satisfação?

Figure C.1: *First Page of the Interview Script in Portuguese*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170
https://www.ime.usp.br

8. Durante a pandemia, onde todos foram obrigados a trabalhar remotamente, o Mob Programming foi benéfico? Pode destacar alguns aspectos? Funcionou bem remotamente? Você prefere presencialmente ou remotamente? Você considera que usar o Mob Programming de forma remota é uma boa opção para o trabalho remoto?
9. Quantas pessoas você considera ideal para programar em grupo usando o Mob Programming?
10. O número ideal na sua opinião de pessoas trabalhando em grupo usando o Mob Programming presencialmente é o mesmo que remotamente? Se não, porque a diferença no tamanho ideal do grupo de desenvolvedores programando juntos remotamente versus presencialmente?
11. Ao usar o Mob Programming remotamente quais softwares vocês usaram? Quais podem ser melhorados? Alguma sugestão de melhoria para os softwares usados?
12. A configuração da sala antes da pandemia como era? Pode ser melhorada?
13. No pós-pandemia você se sente seguro em trabalhar em grupo presencialmente usando o Mob Programming? No pós-pandemia, quando retornarem ao trabalho presencial, vocês pretendem continuar a usar o Mob Programming presencialmente?
14. Ainda sobre o pós-pandemia, vocês vão adotar algumas medidas como divisória de acrílico separando os programadores, uso obrigatório de máscaras, álcool em gel, apresentação do comprovante de duas doses da vacina e testes semanais de COVID? Você gostaria que tais medidas fossem adotadas? Tem alguma sugestão de medida adicional que você gostaria que fosse adotada?
15. Quais foram as dificuldades ao começar a usar o Mob Programming? Quais foram as dificuldades ao começar a usar o Mob Programming remotamente?
16. Comente por favor, especificamente sobre as dificuldades e desafios para o trabalho que a pandemia causou e se na sua opinião usar o Mob Programming ajudou a melhorar ou não a experiência de programação, considerando os aspectos sociais de não estar programando sozinho e sim conectado o dia inteiro com um grupo de programadores (as) usando o Mob Programming.
17. Algum aspecto do Mob Programming a sua equipe aprimorou? Pode ser técnico, relacionado a algum software, comportamento da equipe, rotação, intervalos, padrão de colaboração da equipe, definição de papéis (piloto, co-piloto, pesquisador, tomador de notas, do contra, facilitador, etc).
18. Por favor, comente sobre os pontos positivos e negativos do Mob Programming na sua experiência.
19. Ao usar o Mob Programming em algum momento alguém da sua equipe precisou se ausentar? Se sim, como ficou o desempenho da equipe com uma pessoa ausente?
20. Se a resposta for sim para a pergunta anterior, em algum momento mais de uma pessoa precisou se ausentar? Se sim, como ficou o desempenho da equipe com mais de uma pessoa ausente?

Figure C.2: *Second Page of the Interview Script in Portuguese*

Appendix D

Interview Script in English

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo - SP - Telephone: +55 11 3091-6170 | https://www.ime.usp.br

INTERVIEW SCRIPT

Research Title: Mob Programming: Measures, Effects, and Results
Researcher Name: Herez Moise Kattan
Script Name: Interview on Mob Programming in Complex Tasks and Remote Work
File Name: InterviewScript.pdf

First, thank you very much for your cooperation in participating in the research on Mob Programming. Your practical experience is valuable to deepening our scientific knowledge about this new group programming technique called Mob Programming, and the focus of the study is not only to reduce the cost of software development but also to improve the programming experience for the programmer. Thus, the research is a Grounded Theory about this programming technique called Mob Programming to verify real projects and their benefits in the context of Software Development. Specifically in complex tasks and remote work, both about computer programming. Remember, you can abandon the research without any embarrassment or prejudice.

1. Do you consider that Mob Programming was beneficial in complex tasks? Why?
2. Has the quality of the software improved or not? Has your team measured the improvement or deterioration in the quality of the software product produced using Mob Programming? How was it valued?
3. Are you satisfied with Mob Programming? Do you choose to work solo? Do you prefer pair programming? Depends on the complexity of the task?
4. Did you find Mob Programming more useful for complex tasks than for simple or medium-complexity tasks? Is it a good idea to work entirely on Mob Programming always, or is it better only on complicated tasks?
5. Have you measured team performance or the quality of software produced when developing a complex task versus an easy or medium complexity task using Mob Programming? If so, how did you measure it, and what were the results?
6. Did your team attempt to learn something? Were they together or apart? Was it at the beginning of the Mob Programming session? How was the learning experience? Do you consider that Mob Programming contributed to your learning? What did you learn using Mob Programming?
7. Is it valuable to toggle between Mob Programming and Pair Programming? About knowledge sharing and learning, are they the same or different? Does it generate satisfaction?
8. Under the pandemic, where everyone was far-fetched to work remotely, was Mob Programming beneficial? Can you highlight the positives and negatives? Have you preferred in-person or remote? Is it a good option for remote work to adopt Mob Programming remotely?

Page **1** of **2**

Figure D.1: *First Page of the Interview Script in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo - SP - Telephone: +55 11 3091-6170 | https://www.ime.usp.br

9. How many people are ideal for group programming using Mob Programming?
10. Is the ideal number of people working in groups using Mob Programming in person the same as remotely? Why the difference in the fitting size of the group of developers programming together remotely versus in person?
11. When using Mob Programming remotely, what software did you use? Which ones can be improved? Any suggestions for improvement for the software used?
12. How was the room set up before the pandemic? Can it be improved?
13. In the post-pandemic, do you feel safe working in a group in person using Mob Programming? Post-pandemic, do you intend to continue using Mob Programming face-to-face?
14. Still, on the post-pandemic, will you adopt steps such as an acrylic divider separating programmers, mandatory use of masks, gel alcohol, proof of vaccine, and weekly COVID tests? Would you like such measures to be adopted? Do you have any suggestions for further acts you would like to see taken?
15. What were the difficulties when starting to use Mob Programming? What were the challenges when starting to use Mob Programming remotely?
16. Please comment on the difficulties and challenges to work that the pandemic has caused and if using Mob Programming has helped to improve the programming experience or not, considering the social aspects of not programming alone but connecting the day with a group of programmers using Mob Programming.
17. Has your team improved any aspect of Mob Programming? It can be technical, related to software, team behavior, rotation, intervals, team collaboration pattern, and role definition (pilot, co-pilot, researcher, note taker, contrarian, facilitator, and so on).
18. Please comment on the positives and negatives of Mob Programming in your experience.
19. When using Mob Programming, did someone on your team need to leave at any time? If yes, how was the team's performance with one person absent?
20. If the answer is yes to the previous question, did in your team at any time more than one person need to be away? If yes, how was the team performance' with more than one person absent?

Figure D.2: *Second Page of the Interview Script in English*

Appendix E

TCLE

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170 | https://www.ime.usp.br

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Título da Pesquisa	:	Mob Programming: Medidas, Efeitos e Resultados
Pesquisador Responsável	:	Herez Moise Kattan
Nome do Participante	:	

Você está sendo convidado (a) para ser participante da pesquisa intitulada "Mob Programming: Medidas, Efeitos e Resultados" de responsabilidade do pesquisador Herez Moise Kattan, que se compromete a seguir a Resolução CNS nº 466/2012 e a Resolução CNS nº 510/2016 relacionadas às Pesquisas com Seres Humanos, respeitando todos os seus direitos descritos neste documento.

Leia cuidadosamente todos os itens abaixo e pergunte sobre qualquer dúvida que você tiver. Caso se sinta esclarecido(a) sobre as informações que estão neste Termo e aceite fazer parte do estudo, peço que rubrique todas as páginas e assine ao final deste documento, em duas vias, sendo uma via sua e a outra do pesquisador. Saiba que você tem total direito de não querer participar. Você tem a total liberdade de participar ou deixar de participar do estudo, sem que isso lhe traga algum prejuízo ou risco, podendo interromper sua participação a qualquer momento caso se sinta incomodado(a) com a mesma. O seu nome será mantido em sigilo, sendo que o que disser não lhe resultará em qualquer dano à sua integridade e privacidade.

1. O trabalho tem por objetivo aprofundar o conhecimento científico sobre uma nova técnica de programação de computadores em Grupo chamada Mob Programming e o foco do estudo não é apenas reduzir o custo de desenvolvimento de software, mas também melhorar a experiência de programação para o(a) programador(a). Assim, a pesquisa é uma Teoria Fundamentada nos Dados (Grounded Theory) sobre essa técnica de programação chamada Mob Programming com o objetivo de verificar em projetos reais os seus benefícios no contexto de Desenvolvimento de Software. Especificamente em tarefas complexas e trabalho remoto, ambos sobre programação de computadores.

2. Se você concordar em participar, a sua participação nesta pesquisa consistirá em conceder uma entrevista de até 30 minutos, com a anuência da empresa e/ou chefe, se for o caso. Preencher um questionário online sobre o seu conhecimento em programação de computadores e a sua experiência com a programação em grupo ao redor de um mesmo computador (Mob Programming) durante a realização do seu trabalho de programação em uma tarefa complexa e/ou trabalho remoto. A entrevista, o questionário e também a análise do código fonte produzido estarão relacionados à sua experiência usando Mob Programming para desenvolver um software complexo real do dia-a-dia como colaborador(a) programador(a).

3. A metodologia da pesquisa é Grounded Theory (Teoria Fundamentada nos dados), onde os dados brutos reais coletados durante a sua experiência prática usando a técnica serão analisados para produzir uma teoria fundamentada nestes dados coletados.

4. O local e a infraestrutura onde será usada a programação usando Mob Programming será a da própria empresa onde o(a) colaborador(a) programador(a) trabalha com a anuência da empresa.

Rubrica do pesquisador: _____. Rubrica do participante: _____. Página **1** de **3**

Figure E.1: *First Page of the Termo de Consentimento Livre e Esclarecido (TCLE)*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170 | https://www.ime.usp.br

5. A coleta do áudio da entrevista, fornecimento do questionário e coleta dos dados será de responsabilidade do pesquisador doutorando Herez Moise Kattan. Os projetos desenvolvidos por meio da nova técnica chamada Mob Programming deverão ser preferencialmente de código-fonte aberto e obrigatoriamente produzidos utilizando a técnica de programação Mob Programming. A sua entrevista será gravada e transcrita. O conteúdo das gravações, transcrições e questionários serão de uso específico para essa pesquisa e serão destruídas após 5 anos do término da pesquisa. Os dados serão guardados em arquivos do próprio pesquisador, fora da rede, e estarão sob responsabilidade do pesquisador. Cabe destacar, além dos riscos e benefícios relacionados com a participação na pesquisa, há riscos característicos do ambiente virtual, meios eletrônicos, ou atividades não presenciais, em função das limitações das tecnologias utilizadas. Adicionalmente, há limitações dos pesquisadores para assegurar total confidencialidade e potencial risco de sua violação.

6. De acordo com as orientações contidas nas Resoluções CNS 466/12 (saúde) e 510/16 (humanas e sociais). Analisamos os riscos envolvidos nesta pesquisa e os riscos tangíveis na participação neste estudo são os mesmos envolvidos nas atividades de programação que você como colaborador(a) programador(a) da sua organização já corre e serão os mesmos caso aceite ser voluntário nesta pesquisa. Para mitigar tais riscos, você deve ter uma boa postura ao programar, pés bem apoiados, uma boa cadeira que permita ficar com as costas eretas e o monitor bem posicionado do ponto de vista ergonômico.

7. Embora você provavelmente não terá um benefício direto em participar desta pesquisa, a sua participação será muito valiosa para esta Pesquisa. Você estará contribuindo com a Ciência, pois existirão publicações compartilhando o conhecimento adquirido sobre a nova técnica de programação Mob Programming. Se realmente essa nova técnica levar a uma melhor experiência de programação para o(a) programador(a) e caso vocês na sua empresa resolverem adotar a técnica, você poderá se beneficiar em programar em grupo usando essa nova técnica ao invés de programar sozinho(a) como normalmente acontece. Portanto, o principal benefício com a sua participação nesta pesquisa será que o resultado da pesquisa leve a um conhecimento aprofundado para os profissionais e comunidade científica sobre a técnica Mob Programming em tarefas complexas e trabalho remoto.

8. Você tem o Direito ao ressarcimento de despesas relativas à participação na pesquisa. Você tem Direito a indenização, caso sinta-se prejudicado pela participação na pesquisa. Os participantes não terão nenhuma despesa ao participar da pesquisa e poderão retirar sua concordância na continuidade da pesquisa a qualquer momento. Tem direito ao acompanhamento e assistência imediata, integral e gratuita (durante, após e/ou interrupção da pesquisa). Você tem garantia de receber uma resposta a alguma dúvida durante ou após as entrevistas. Você terá acesso gratuito as publicações e a tese de doutorado.

9. Você tem o Direito a assistência gratuita e a buscar indenização. Não há nenhum valor econômico a receber ou a pagar aos voluntários pela participação e não haverá qualquer despesa decorrente desta participação portanto não haverá necessidade de ressarcimento pelos pesquisadores. Caso ocorra algum dano decorrente da participação no estudo, os voluntários poderão pleitear indenização, segundo as determinações do Código Civil (Lei nº 10.406 de 2002) e das Resoluções 466/12 e 510/16 do Conselho Nacional de Saúde.

Figure E.2: *Second Page of the Termo de Consentimento Livre e Esclarecido (TCLE)*

INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO



IME-USP

Rua do Matão, 1010 – CEP 05508-090 – São Paulo – SP – Telefone: +55 11 3091-6170 | https://www.ime.usp.br

10. O nome dos participantes será mantido em sigilo, assegurando assim a sua privacidade. Os participantes terão livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências, enfim, tudo o que queiram saber antes, durante e depois da sua participação.

11. Os dados coletados serão utilizados única e exclusivamente, para fins desta pesquisa, e os resultados poderão ser apresentados em reuniões e ou publicações (revistas, artigos, tese de doutorado, jornais científicos e de circulação), contudo, sua identidade não será revelada durante essas apresentações.

12. O estudo não acarretará nenhuma despesa para o participante. Além disso, sei que posso abandonar minha participação na pesquisa quando quiser e que não receberei nenhum pagamento por esta participação. É importante que o participante guarde em seus arquivos uma via deste documento e é garantido o envio da via assinada pelo pesquisador ao participante da pesquisa.

Declaro que, após convenientemente esclarecido pelo pesquisador e ter entendido o que me foi explicado, consinto em participar do presente Projeto de Pesquisa e estou ciente que existem duas vias deste Termo e que ficarei com uma delas.

Qualquer dúvida, entre em contato com o pesquisador Herez Moise Kattan pelo telefone: +55 11 96100-2585, e-mail: herez@ime.usp.br e/ou com a Comissão de Pesquisa do IME-USP, telefone: +55 11 3091-6170, e-mail: cpq@ime.usp.br, atendimento presencial na sala B11, bloco B do IME-USP e/ou com a Comissão Nacional de Ética em Pesquisa-CONEP, telefone: (61) 3315-5877, e-mail: conep@saude.gov.br e/ou Comitê de Ética em Pesquisa (CEP) da Escola de Artes, Ciências e Humanidades da Universidade de São Paulo (EACH-USP), situado à Rua Arlindo Bettio, 1000, Prédio I1, Sala T14, Vila Guaraciaba, São Paulo, SP, CEP 03828-000; telefone: (11) 3091-1046; email: cep-each@usp.br; horário de funcionamento: Segundas às sextas-feiras: das 09:00 às 11:00 e das 14:00 às 16:00 h.

Eu, _____ declaro ter sido informado e concordo em ser participante do Projeto de pesquisa acima descrito.

_____ , _____ de _____ de 20_____.

_____
Assinatura do Participante da Pesquisa

_____
Herez Moise Kattan (Pesquisador)

Figure E.3: *Third Page of the Termo de Consentimento Livre e Esclarecido (TCLE)*

Appendix F

Informed Consent Document in English

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo - SP - Telephone: +55 11 3091-6170 | https://www.ime.usp.br

INFORMED CONSENT DOCUMENT

Research Title: Mob Programming: Measures, Effects, and Results
Researcher Name: Herez Moise Kattan
Participant Name:

You are requested to participate in the research entitled "Mob Programming: Measures, Effects, and Results" under the responsibility of the researcher Herez Moise Kattan, who declares to know and comply with the Brazilian Ethical Resolutions with the National Commission of Ethics in Research (CONEP), including the guidelines of Brazilian National Council of Health Resolution Number (CNS) 466/2012 and Brazilian National Council of Health Resolution Number (CNS) 510/2016, respecting all voluntary participants rights. This research also complies with the United States federal regulations subpart A of 45 Code of Federal Regulations (CFR) part 46, often referred to as the "Common Rule" or "Protection of Human Subjects Regulations." The Federal regulations 45 CFR 46.116 from the United States provide the framework for the type of information that must be part of the consent process.

Please read the items below carefully and ask any questions you may have. Supposing you feel clarified about the information in this Term and accept to be part of the study, please initial all pages and sign at the end of this document in two copies, one for you and the other for the researcher. Know you have full right not to want to participate. You are free to participate or not participate in the study without causing any harm or risk. You can stop your participation on any occasion if you feel uncomfortable with it. Your name will be kept confidential, and your say will not spoil your integrity and privacy.

1. The work aims to deepen the scientific knowledge about a new technique of group computer programming called Mob Programming, and the focus of the study is not only to reduce the cost of software development but also to improve the programming experience for the programmer. Thus, the research is a Grounded Theory about this programming technique called Mob Programming to verify in real projects its benefits in the context of Software Development. Specifically in complex tasks and remote work, both about computer programming.

2. In case you agree to take part. Your participation in this research will consist of granting an interview of up to 30 minutes, with company and boss consent' if applicable. Complete an online questionnaire about your knowledge of computer programming and your experience with Mob Programming while performing your programming work on a complex task or remote work. The interview, the questionnaire, and the analysis of the source code produced will be related to your experience using Mob Programming to develop complex day-to-day software as a professional developer.

3. The research method is Grounded Theory. The raw data collected will be analyzed to produce a theory grounded in these collected data about your practical experience using the technique.

4. The location and infrastructure for the programming using Mob Programming are the company's own, where the programmer works with the company's consent.

Researcher initials signature: _____. Participant initials signature: _____. Page **1** of **3**

Figure F.1: *First Page of the Informed Consent Document in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO

IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo - SP - Telephone: +55 11 3091-6170 | https://www.ime.usp.br

5. The collection of the audio of the interview, supply of the questionnaire, and data collection will be the responsibility of the doctoral researcher Herez Moise Kattan. Projects developed using Mob Programming should be produced using the Mob Programming technique. Your interview will record and transcribed. The content of the recordings, transcripts, and questionnaires will be of specific use for this research and will destroy after five years from the end of the study. The data will be stored in the researcher's files, outside the network, and will be under the researcher's responsibility. The risks and benefits related to participation in the research dwell on the dangers of the virtual environment, electronic media, or non-face-to-face activities due to the limitations of the technologies used. Existing barriers for researchers to ensure complete confidentiality and the potential risk of breach or leak.

6. By the guidelines of CNS Resolutions 466/12 (health) and 510/16 (human and social). We analyzed the risks involved in this research. The risks involved in participating in this study are the same implicated in the programming activities you face as a professional programmer and collaborator in your organization. Moreover, these risks are the same if you accept to act as a volunteer in this research. To mitigate such risks, you must have good posture when programming, feet well supported, a good chair that allows you to stand upright, and the monitor well positioned from an ergonomic point of view.

7. Although you will likely not have a direct benefit from participating in this research, your participation will be valuable to this research. You will contribute to Science, as there will be publications sharing the knowledge acquired about the new programming technique Mob Programming. If this new technique leads to a better programming experience for the programmer and if you in your company decide to adopt the approach, you can benefit from programming in a group using this new technique instead of programming alone as usual. Therefore, the main benefit of your participation in this research will be that the result of the study will lead to in-depth knowledge for professionals and the scientific community about the Mob Programming technique in complex tasks and remote work.

8. You have the right to reimbursement of the expenses related to research participation. Participants will not incur any costs when participating in the research and may withdraw their agreement to continue the study at any time. You are guaranteed to receive an answer to any queries during or after the interviews. You will have free access to publications and the doctoral thesis.

9. You have the right to free assistance and to seek compensation. There is no economic value to be received or paid to volunteers for their participation, and there will be no expenses arising from it. Thus, there will be no need for reimbursement by the researchers.

10. The name of the participants will be kept confidential, thus ensuring their privacy. Participants will have free access to additional information and clarifications about the study and its consequences, in short, everything they want to know before, during, and after their participation.
11. The data collected will serve exclusively for this research. The results shall appear at meetings, in magazines, articles, doctoral dissertations, circulation journals, or scientific. However, during these presentations, your identity will not get out.

Figure F.2: *Second Page of the Informed Consent Document in English*

INSTITUTE OF MATHEMATICS AND STATISTICS OF UNIVERSITY OF SAO PAULO



IME-USP

Matao Street, 1010 - CEP 05508-090 - Sao Paulo - SP - Telephone: +55 11 3091-6170 | https://www.ime.usp.br

12. The study will not incur any expense to the participant. In addition, I know that I can withdraw from participating in the research at any time and that I will not receive any payment for this participation. The participant must keep a copy of this document in their files, and the copy signed by the researcher is warranted and will send to the research participant.

I declare that I accept to be a volunteer after being explained by the researcher. I have understood the explanation. I give my consent to participate in this Research Project. I know that there are two copies of this Term and that I will keep one of them.

If you have any questions, contact Herez Moise Kattan by telephone at (+55 11) 96100-2585, email: herez@ime.usp.br or with the Research Commission of IME-USP, telephone: (+55 11) 3091-6170, email: cpq@ime.usp.br, face-to-face service in room B11, block B of IME-USP or with the National Research Ethics Commission-CONEP, telephone: (+55 61) 3315-5877, email: conep@saude.gov.br or Research Ethics Committee (CEP) of the School of Arts, Sciences and Humanities of the University of Sao Paulo (EACH-USP), located at Rua Arlindo Bettio, 1000, Building I1, Room T14, Vila Guaraciaba, Sao Paulo, SP, CEP 03828-000; telephone: (+55 11) 3091-1046; email: cep-each@usp.br; opening hours: Mondays to Fridays from 09:00 to 11:00 and from 14:00 to 16:00.

I, _____, declare that I have been informed and agree to be a participant in the Research Project described above.

_____ , _____ of _____ of 2022.

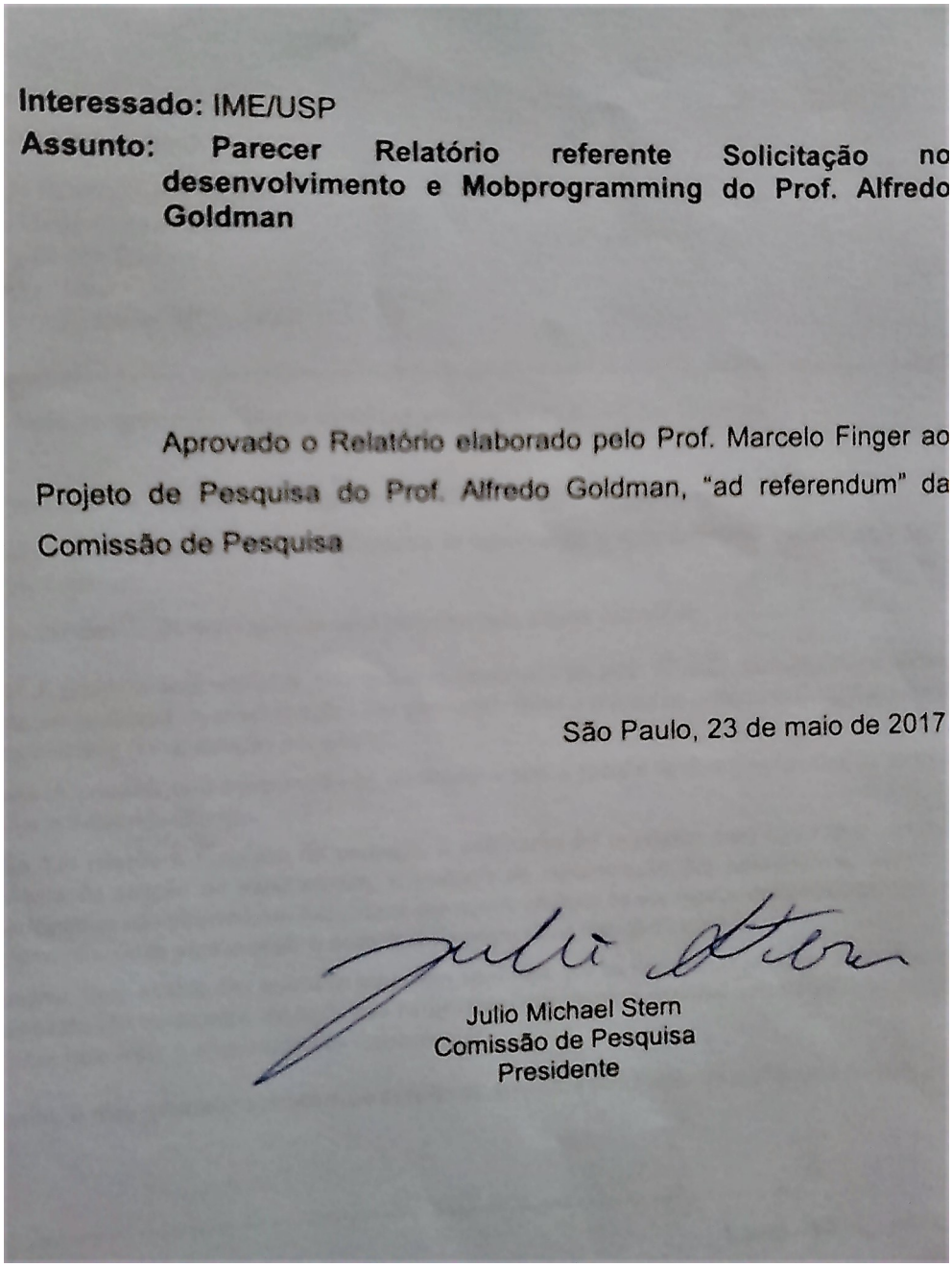
_____
Research Participant Signature

_____
Herez Moise Kattan (Researcher)

Figure F.3: *Third Page of the Informed Consent Document in English*

Appendix G

Authorizations of the Ethics Committees



Interessado: IME/USP

Assunto: **Parecer Relatório referente Solicitação no desenvolvimento e Mobprogramming do Prof. Alfredo Goldman**

Aprovado o Relatório elaborado pelo Prof. Marcelo Finger ao Projeto de Pesquisa do Prof. Alfredo Goldman, "ad referendum" da Comissão de Pesquisa

São Paulo, 23 de maio de 2017

Julio Michael Stern
Comissão de Pesquisa
Presidente

Figure G.1: *Authorization of the Ethics Committee of the Institute of Mathematics and Statistics of the University of Sao Paulo.*

Flávio Soares Corrêa da Silva
fcs@ime.usp.br

Dezembro 2018, São Paulo - Brazil

Solicitação de autorização ao Comitê de Pesquisa do IME-USP

O Herez Moise Kattan, aluno do doutorado sob minha supervisão, está trabalhando em uma pesquisa onde são estudadas novas formas de programação. Essa pesquisa já foi aprovada pelo comitê anteriormente e tem um artigo publicado no WBMA 2017. Como a pesquisa envolve a realização de entrevista com alguns alunos participantes da disciplina LAB XP 2018 e Laboratório Avançado de Métodos Ágeis de Desenvolvimento de Software 2018, solicitamos a aprovação para a sua realização e futura publicação.

A entrevista é sobre a experiência dos alunos com Mob Programming na disciplina de LAB XP e Laboratório Avançado de Métodos Ágeis de Desenvolvimento de Software. Mob Programming é uma prática ágil relacionada com XP, é sobre programação em grupo, similar à programação em par, porém, ao invés de duplas, é feita por um grupo de pessoas simultaneamente. Seguem abaixo alguns aspectos ligados à pesquisa.

Critérios de seleção de participantes: Estamos procurando participantes maiores de 18 anos que saibam programar e esteja desenvolvendo algum software por meio da prática de Mob Programming. Esperamos contar com os participantes de alguns alunos da disciplina Lab XP que aderiram ao Mob Programming neste estudo.

Procedimentos: Caso você concorde em participar do experimento, solicitamos que você participe de uma entrevista sobre seu conhecimento em programação e sua experiência com Mob Programming durante o Lab XP e Laboratório Avançado de Métodos Ágeis de Desenvolvimento de Software.

Riscos e Desconfortos: Os riscos envolvendo sua participação neste estudo não são maiores do que aqueles existentes nas suas atividades regulares de programação.

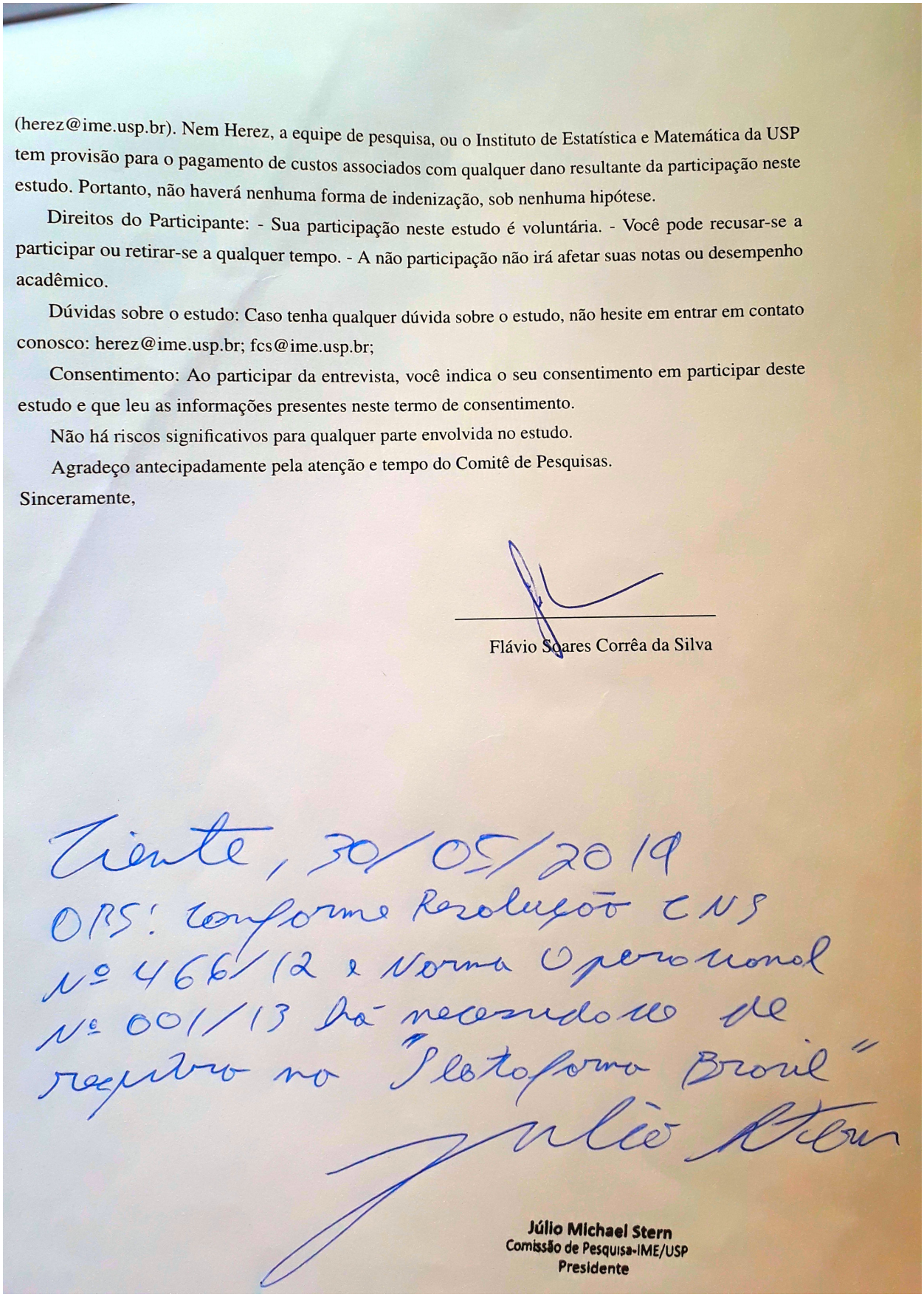
Benefícios: Sua participação será muito valiosa para nossa pesquisa. Você provavelmente não terá benefícios diretos ao participar deste estudo. No entanto, a Publicação com os resultados experimento estará disponível. Esperamos que os resultados do experimento levem a uma melhor compreensão sobre o Mob Programming.

Compensações e custos: Não há compensação pela participação neste estudo. Não haverá custos, além do seu tempo, pela sua participação no estudo.

Anonimato, Confidencialidade e Privacidade: Os seguintes procedimentos irão garantir seu anonimato, privacidade e confidencialidade. Na entrevista, não será solicitada nenhuma informação pessoal. Assim, as questões sobre sua experiência em desenvolvimento não serão vinculadas a você. Para garantir que esta pesquisa está sendo realizada da forma correta, o Comitê de Pesquisa do Instituto de Matemática e Estatística da Universidade de São Paulo poderá revisar os registros deste estudo.

Em caso de dano/prejuízo: Caso você tenha qualquer problema, dano ou desconforto como resultado da participação neste estudo, por favor, entre em contato com Herez Moise Kattan por e-mail

Figure G.2: *Solicitation to the Research Committee of the Institute of Mathematics and Statistics of the University of Sao Paulo.*



(herez@ime.usp.br). Nem Herez, a equipe de pesquisa, ou o Instituto de Estatística e Matemática da USP tem provisão para o pagamento de custos associados com qualquer dano resultante da participação neste estudo. Portanto, não haverá nenhuma forma de indenização, sob nenhuma hipótese.

Direitos do Participante: - Sua participação neste estudo é voluntária. - Você pode recusar-se a participar ou retirar-se a qualquer tempo. - A não participação não irá afetar suas notas ou desempenho acadêmico.

Dúvidas sobre o estudo: Caso tenha qualquer dúvida sobre o estudo, não hesite em entrar em contato conosco: herez@ime.usp.br; fcs@ime.usp.br;

Consentimento: Ao participar da entrevista, você indica o seu consentimento em participar deste estudo e que leu as informações presentes neste termo de consentimento.

Não há riscos significativos para qualquer parte envolvida no estudo.

Agradeço antecipadamente pela atenção e tempo do Comitê de Pesquisas.

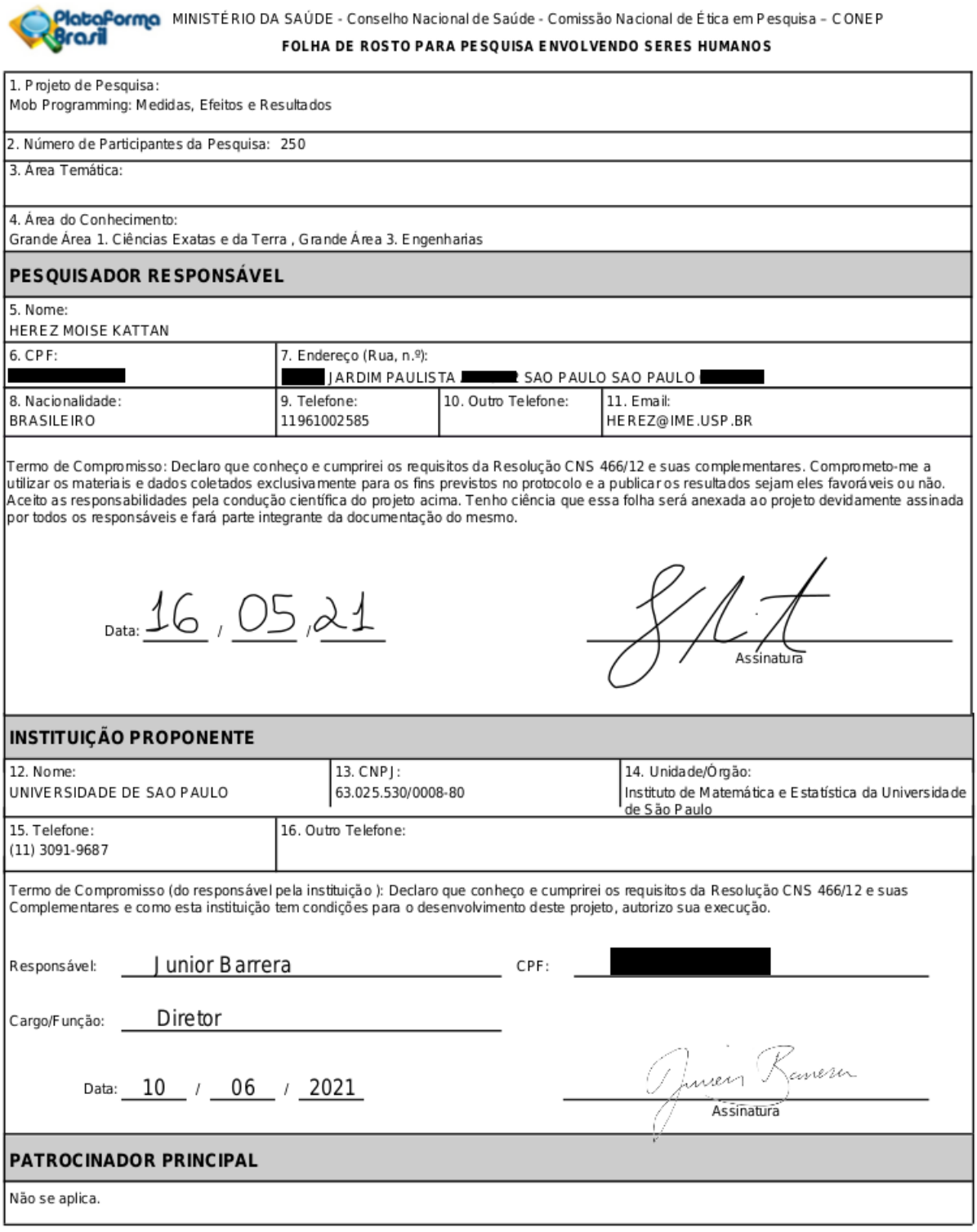
Sinceramente,

Flávio Soares Corrêa da Silva

Ciente, 30/05/2019
ORS: conforme Resolução CNS nº 466/12 e Norma Operacional nº 001/13 há necessidade de registro na "Plataforma Brasil"

Júlio Michael Stern
Comissão de Pesquisa-IME/USP
Presidente

Figure G.3: *Authorization from the IME-USP Research Committee and need to register in the Plataforma Brasil.*



MINISTÉRIO DA SAÚDE - Conselho Nacional de Saúde - Comissão Nacional de Ética em Pesquisa – CONEP

FOLHA DE ROSTO PARA PESQUISA ENVOLVENDO SERES HUMANOS

1. Projeto de Pesquisa:
Mob Programming: Medidas, Efeitos e Resultados

2. Número de Participantes da Pesquisa: 250

3. Área Temática:

4. Área do Conhecimento:
Grande Área 1. Ciências Exatas e da Terra , Grande Área 3. Engenharias

PESQUISADOR RESPONSÁVEL

5. Nome:
HEREZ MOISE KATTAN

6. CPF:	7. Endereço (Rua, n.º):		
[redacted]	[redacted] JARDIM PAULISTA [redacted] SAO PAULO SAO PAULO [redacted]		
8. Nacionalidade: BRASILEIRO	9. Telefone: 11961002585	10. Outro Telefone:	11. Email: HEREZ@IME.USP.BR

Termo de Compromisso: Declaro que conheço e cumprirei os requisitos da Resolução CNS 466/12 e suas complementares. Comprometo-me a utilizar os materiais e dados coletados exclusivamente para os fins previstos no protocolo e a publicar os resultados sejam eles favoráveis ou não. Aceito as responsabilidades pela condução científica do projeto acima. Tenho ciência que essa folha será anexada ao projeto devidamente assinada por todos os responsáveis e fará parte integrante da documentação do mesmo.

Data: 16 / 05 / 21

Assinatura

INSTITUIÇÃO PROPONENTE

12. Nome:	13. CNPJ:	14. Unidade/Órgão:
UNIVERSIDADE DE SAO PAULO	63.025.530/0008-80	Instituto de Matemática e Estatística da Universidade de São Paulo

15. Telefone:	16. Outro Telefone:
(11) 3091-9687	

Termo de Compromisso (do responsável pela instituição): Declaro que conheço e cumprirei os requisitos da Resolução CNS 466/12 e suas Complementares e como esta instituição tem condições para o desenvolvimento deste projeto, autorizo sua execução.

Responsável: Junior Barrera CPF: [redacted]

Cargo/Função: Diretor

Data: 10 / 06 / 2021

Assinatura

PATROCINADOR PRINCIPAL

Não se aplica.

Figure G.4: *The cover sheet of this research is in the registry of Plataforma Brasil of the National Commission of Ethics in Research (CONEP).*

USP - ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES DA UNIVERSIDADE DE SÃO PAULO - EACH/USP

Plataforma Brasil

PARECER CONSUBSTANCIADO DO CEP

DADOS DO PROJETO DE PESQUISA

Título da Pesquisa: Mob Programming: Medidas, Efeitos e Resultados
Pesquisador: HEREZ MOISE KATTAN
Área Temática:
Versão: 1
CAAE: 61707421.0.0000.5390
Instituição Proponente: Instituto de Matemática e Estatística da Universidade de São Paulo
Patrocinador Principal: Financiamento Próprio

DADOS DO PARECER

Número do Parecer: 5.639.128

Apresentação do Projeto:
Trata-se da análise do projeto de pesquisa "Mob Programming: Medidas, Efeitos e Resultados." Tramita em versão 1 submetido ao CEP em 16/08/2022. É um projeto de doutorado em Ciência da Computação do INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO. Tem como pesquisador responsável e executante Herez Moise Kattan. A pesquisa aborda a prática emergente, popular e atual para aumentar a qualidade no desenvolvimento de software, aprendizagem e satisfação dos programadores: Mob Programming. Será realizada com 200 participantes (programadores) por meio de entrevista e resposta a um questionário. Tem autrização para realização da pesquisa pela Rocket.Chat. A técnica estudada é chamada de Mob Programming, onde todos programam juntos em um único computador se for presencialmente e se for remotamente todos compartilham a mesma tela e programam juntos. O método de pesquisa é Grounded Theory, onde serão conduzidas entrevistas, questionários e observações que produzem dados brutos sobre essas observações. Então, são analisados esses dados por meio de open code, constant comparison method, core category, selective coding, memoing, sorting e, theoretical coding seguindo o método de pesquisa Grounded Theory para servir como recomendação para desenvolvedores de software com sugestões sobre como usar o Mob Programming em tarefas complexas e trabalho remoto.

Objetivo da Pesquisa:
Objetivo Primário:

Endereço: Av. Arlindo Béttio, nº 1000
Bairro: Ermelino Matarazzo **CEP:** 03.828-000
UF: SP **Município:** SAO PAULO
Telefone: (11)3091-1046 **E-mail:** cep-each@usp.br

Figure G.5: *Page 1 of the Written Approval of the Plataforma Brasil of the National Commission of Ethics in Research (CONEP).*

USP - ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES DA UNIVERSIDADE DE SÃO PAULO - EACH/USP

Plataforma Brasil

Continuação do Parecer: 5.639.128

Investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada Mob Programming.

Objetivo Secundário:
Usar Mob Programming remotamente preserva os benefícios de usar a técnica presencialmente

Avaliação dos Riscos e Benefícios:

Riscos:
De acordo com as orientações contidas nas Resoluções CNS 466/12 (saúde) e 510/16 (humanas e sociais). O autor analisou os riscos envolvidos nesta pesquisa e os riscos tangíveis na participação neste estudo são os mesmos inerentes às atividades de programação que o participante como colaborador(a) programador(a) da sua organização já corre e serão os mesmos caso aceite ser voluntário nesta pesquisa. Para mitigar tais riscos, sugere no Termo de Consentimento Livre e Esclarecido se esforçar ter uma boa postura ao programar, pés bem apoiados, uma boa cadeira que permita ficar com as costas eretas e o monitor bem posicionado do ponto de vista ergonômico. Adicionalmente afirma que não há possibilidade de danos na dimensão física, psíquica, moral, intelectual, social, cultural ou espiritual do ser humano participante desta pesquisa, em qualquer fase da minha pesquisa ou dela decorrente. Pelo fato de serem apenas observações, entrevistas e questionários bem simples sobre a técnica Mob Programming, nada que cause desconforto ou constrangimento, pois não tem nenhuma pergunta pessoal, são somente perguntas técnicas objetivas sobre a nova técnica que a equipe de profissionais desenvolvimento de software decidiu por conta própria usar e voluntariamente resolveram contribuir com a ciência participando voluntariamente da pesquisa.

Benefícios:
Usar software de boa qualidade com o menor custo e menor número de defeitos é um benefício para os cidadãos e toda a sociedade. Para as organizações desenvolvedoras de software e seus profissionais os principais benefícios são: 1. Descobrir se tarefas complexas de programação de computadores são feitas com maior qualidade em grupo (Mob Programming); 2. Descobrir se os programadores(as) preferem trabalhar usando o Mob Programming, se aumenta a satisfação e aprendizados dos programadores(as); 3. Descobrir se é uma maneira benéfica para desenvolvimento de software remoto

Endereço: Av. Arlindo Béttio, nº 1000
Bairro: Ermelino Matarazzo **CEP:** 03.828-000
UF: SP **Município:** SAO PAULO
Telefone: (11)3091-1046 **E-mail:** cep-each@usp.br

Figure G.6: *Page 2 of the Written Approval of the Plataforma Brasil of the National Commission of Ethics in Research (CONEP).*

USP - ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES DA UNIVERSIDADE DE SÃO PAULO - EACH/USP

Plataforma Brasil

Continuação do Parecer: 5.639.128

Comentários e Considerações sobre a Pesquisa:

Estudo tem temática relevante, uma vez que as tarefas complexas são feitas com mais qualidade usando o Mob Programming. Traz a hipótese de que o Mob Programming aumenta o aprendizado e a satisfação dos programadores. Apresenta uma proposta metodológica pertinente aos objetivos propostos.

Considerações sobre os Termos de apresentação obrigatória:

Vide campo "conclusões ou Pendências e Lista de inadequações"

Recomendações:

Vide campo "conclusões ou Pendências e Lista de inadequações"

Conclusões ou Pendências e Lista de Inadequações:

Trata-se da análise de projeto de pesquisa de doutorado do INSTITUTO DE MATEMÁTICA E ESTATÍSTICA DA UNIVERSIDADE DE SÃO PAULO, sob número: 61707421.0.0000.5390 submetido em 16/08/22.

Considera-se projeto aprovado, uma vez que os termos apresentados estão de acordo com a Resolução nº 466/2012 e a Resolução CNS nº 510/2016 relacionadas às Pesquisas com Seres Humanos do Conselho Nacional de Saúde - Ministério da Saúde.

Como as entrevistas poderão ser realizadas remotamente, recomenda-se, uma vez concluída a coleta dos dados, o pesquisador responsável deverá fazer o download dos mesmos para um dispositivo eletrônico local, apagando todo e qualquer registro em qualquer plataforma virtual, ambiente compartilhado ou nuvem.

Considerações Finais a critério do CEP:

Projeto aprovado, pois está de acordo com a Resolução CNS Nº 510/2016 relacionada à Ética em Pesquisa com Seres Humanos do Conselho Nacional de Saúde do Ministério da Saúde.

Ressalta-se que cabe ao pesquisador responsável encaminhar os relatórios parciais e finais da pesquisa, por meio da Plataforma Brasil, via notificação do tipo "relatório" para que sejam devidamente apreciadas pelo CEP, conforme Norma Operacional CNS n 001/13, item XI.2.d.

Este parecer foi elaborado baseado nos documentos abaixo relacionados:

Tipo Documento	Arquivo	Postagem	Autor	Situação
Outros	caKatan.pdf	13/09/2022	Beatriz Aparecida	Aceito

Endereço: Av. Arlindo Béttio, nº 1000
Bairro: Ermelino Matarazzo **CEP:** 03.828-000
UF: SP **Município:** SAO PAULO
Telefone: (11)3091-1046 **E-mail:** cep-each@usp.br

Figure G.7: *Page 3 of the Written Approval of the Plataforma Brasil of the National Commission of Ethics in Research (CONEP).*

USP - ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES DA UNIVERSIDADE DE SÃO PAULO - EACH/USP

Plataforma Brasil

Continuação do Parecer: 5.639.128

Outros	caKatan.pdf	12:46:43	Ozello Gutierrez	Aceito
Informações Básicas do Projeto	PB_INFORMAÇOES_BASICAS_DO_PROJETO_1753982.pdf	16/08/2022 23:34:49		Aceito
Projeto Detalhado / Brochura Investigador	ProjetoDetalhado_MobProgramming.pdf	16/08/2022 23:30:24	HEREZ MOISE KATTAN	Aceito
TCLE / Termos de Assentimento / Justificativa de Ausência	termo_consentimento_TCLE_Mob_Programming.pdf	16/08/2022 23:26:55	HEREZ MOISE KATTAN	Aceito
Declaração de concordância	Autorizacao_do_RocketChat_para_coleta_de_dados_e_participacao_em_doutorado_sobre_MobProgramming_TERMO_DE_ANUENCIA.pdf	16/08/2022 23:21:09	HEREZ MOISE KATTAN	Aceito
Outros	RoteiroEntrevista.pdf	23/08/2021 14:55:06	HEREZ MOISE KATTAN	Aceito
Outros	Questionario.pdf	23/08/2021 14:53:24	HEREZ MOISE KATTAN	Aceito
Outros	Autorizacao_Imagem_Voz.pdf	23/08/2021 14:51:03	HEREZ MOISE KATTAN	Aceito
Folha de Rosto	folhaDeRosto_PlataformaBrasil_signed.pdf	21/06/2021 16:57:04	HEREZ MOISE KATTAN	Aceito

Situação do Parecer:
Aprovado

Necessita Apreciação da CONEP:
Não

SAO PAULO, 13 de Setembro de 2022

Assinado por:
Beatriz Aparecida Ozello Gutierrez
(Coordenador(a))

Endereço: Av. Arlindo Béttio, nº 1000
Bairro: Ermelino Matarazzo **CEP:** 03.828-000
UF: SP **Município:** SAO PAULO
Telefone: (11)3091-1046 **E-mail:** cep-each@usp.br

Figure G.8: *Page 4 of the Written Approval of the Plataforma Brasil of the National Commission of Ethics in Research (CONEP).*

Appendix H

Institution Official Consent Terms

H.1 Written Approval of the Rocket.Chat



TERMO DE ANUÊNCIA

Declaramos para os devidos fins que estamos de acordo com a execução do projeto de pesquisa intitulado "Mob Programming: Medidas, Efeitos e Resultados", sob a coordenação e a responsabilidade do pesquisador Herez Moise Kattan, mestre em Engenharia de Computação/Software e doutorando em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. O objetivo deste projeto de pesquisa é investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada Mob Programming. Objetivo Específico Primário: Tarefas complexas são resolvidas com maior qualidade usando a técnica Mob Programming de programação em grupo e há benefícios usando essa nova técnica. Objetivo Específico Secundário: Usar Mob Programming remotamente preserva os benefícios de usar a técnica presencialmente.

Assumimos o compromisso de permitir e apoiar o desenvolvimento da referida pesquisa a ser realizada nessa instituição. Declaramos conhecer e cumprir as Resoluções Éticas Brasileiras junto à Comissão Nacional de Ética em Pesquisa (CONEP), em especial as diretrizes da Resolução CNS nº 466/2012 e da Resolução CNS nº 510/2016.

Porto Alegre, 28 de julho de 22.

Gabriel Engel – CEO

Figure H.1: *Autorization of the Rocket.Chat*

H.2 Written Approval of the Brazilian Air Force

MINISTÉRIO DA DEFESA
COMANDO DA AERONÁUTICA
CENTRO DE COMPUTAÇÃO DA AERONÁUTICA DE SÃO JOSÉ DOS CAMPOS

TERMO DE ANUÊNCIA

Declaramos para os devidos fins que estamos de acordo com a execução do projeto de pesquisa intitulado ***Mob Programming*: Medidas, Efeitos e Resultados**, sob a coordenação e a responsabilidade do pesquisador Hercz Moise Kattan, mestre em Engenharia de Computação/Software e doutorando em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. O objetivo deste projeto de pesquisa é investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada *Mob Programming*. Objetivo Específico Primário: Tarefas complexas são resolvidas com maior qualidade usando a técnica *Mob Programming* de programação em grupo e há benefícios usando essa nova técnica. Objetivo Específico Secundário: Usar *Mob Programming* remotamente preserva os benefícios de usar a técnica presencialmente.

Assumimos o compromisso de permitir e apoiar o desenvolvimento da referida pesquisa a ser realizada nesta instituição. Revisaremos o texto que será publicado sobre o Centro de Computação da Aeronáutica de São José dos Campos (CCA-SJ). Declaramos conhecer e cumprir as Resoluções Éticas Brasileiras junto à Comissão Nacional de Ética em Pesquisa (CONEP), em especial as diretrizes da Resolução CNS nº 466/2012 e da Resolução CNS nº 510/2016.

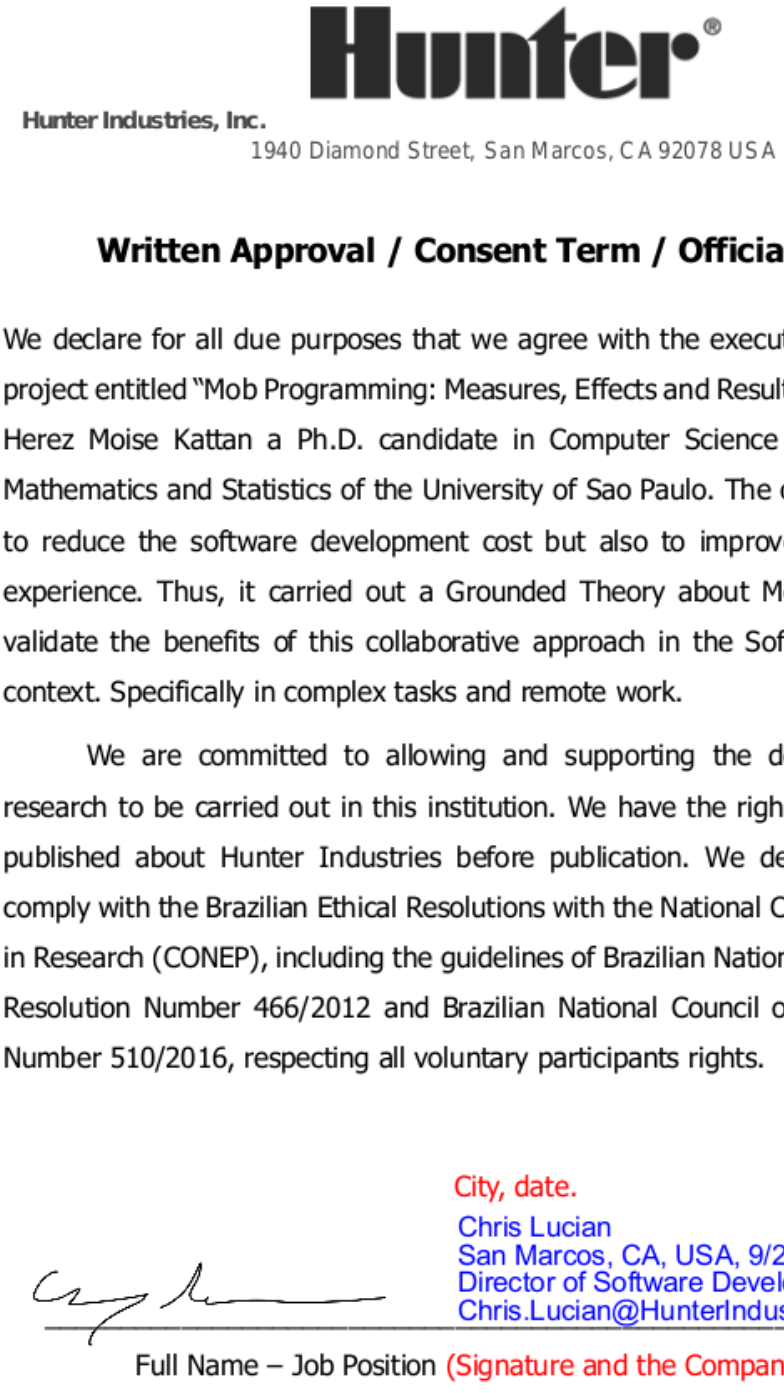
COMANDO DA AERONÁUTICA
CCA SJ

São José dos Campos, 19 de setembro de 2022.

PITERSON MARQUES LISBOA Coronel-Aviador
Chefe do CCA-SJ

Figure H.2: *Autorization of the Brazilian Air Force*

H.3 Written Approval of the Hunter Industries



Hunter Industries, Inc.
1940 Diamond Street, San Marcos, CA 92078 USA

Written Approval / Consent Term / Official Consent

We declare for all due purposes that we agree with the execution of the research project entitled "Mob Programming: Measures, Effects and Results" of the researcher Herez Moise Kattan a Ph.D. candidate in Computer Science at the Institute of Mathematics and Statistics of the University of Sao Paulo. The objective is not only to reduce the software development cost but also to improve the programming experience. Thus, it carried out a Grounded Theory about Mob Programming to validate the benefits of this collaborative approach in the Software Development context. Specifically in complex tasks and remote work.

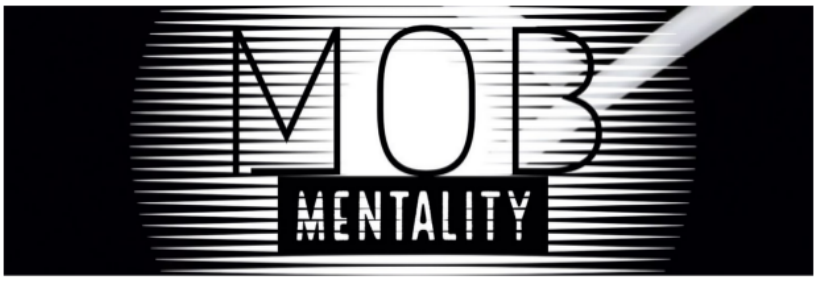
We are committed to allowing and supporting the development of this research to be carried out in this institution. We have the right to review the text published about Hunter Industries before publication. We declare to know and comply with the Brazilian Ethical Resolutions with the National Commission of Ethics in Research (CONEP), including the guidelines of Brazilian National Council of Health Resolution Number 466/2012 and Brazilian National Council of Health Resolution Number 510/2016, respecting all voluntary participants rights.

City, date.
Chris Lucian
San Marcos, CA, USA, 9/23/2022
Director of Software Development
Chris.Lucian@HunterIndustries.com

Full Name – Job Position (Signature and the Company Stamp)

Figure H.3: *Autorization of the Hunter Industries*

H.4 Written Approval of the Mob Mentality



Written Approval / Consent Term / Official Consent

We declare for all due purposes that we agree with the execution of the research project entitled "Mob Programming: Measures, Effects and Results" of the researcher Herez Moise Kattan a Ph.D. candidate in Computer Science at the Institute of Mathematics and Statistics of the University of Sao Paulo. The objective is not only to reduce the software development cost but also to improve the programming experience. Thus, it carried out a Grounded Theory about Mob Programming to validate the benefits of this collaborative approach in the Software Development context. Specifically in complex tasks and remote work.

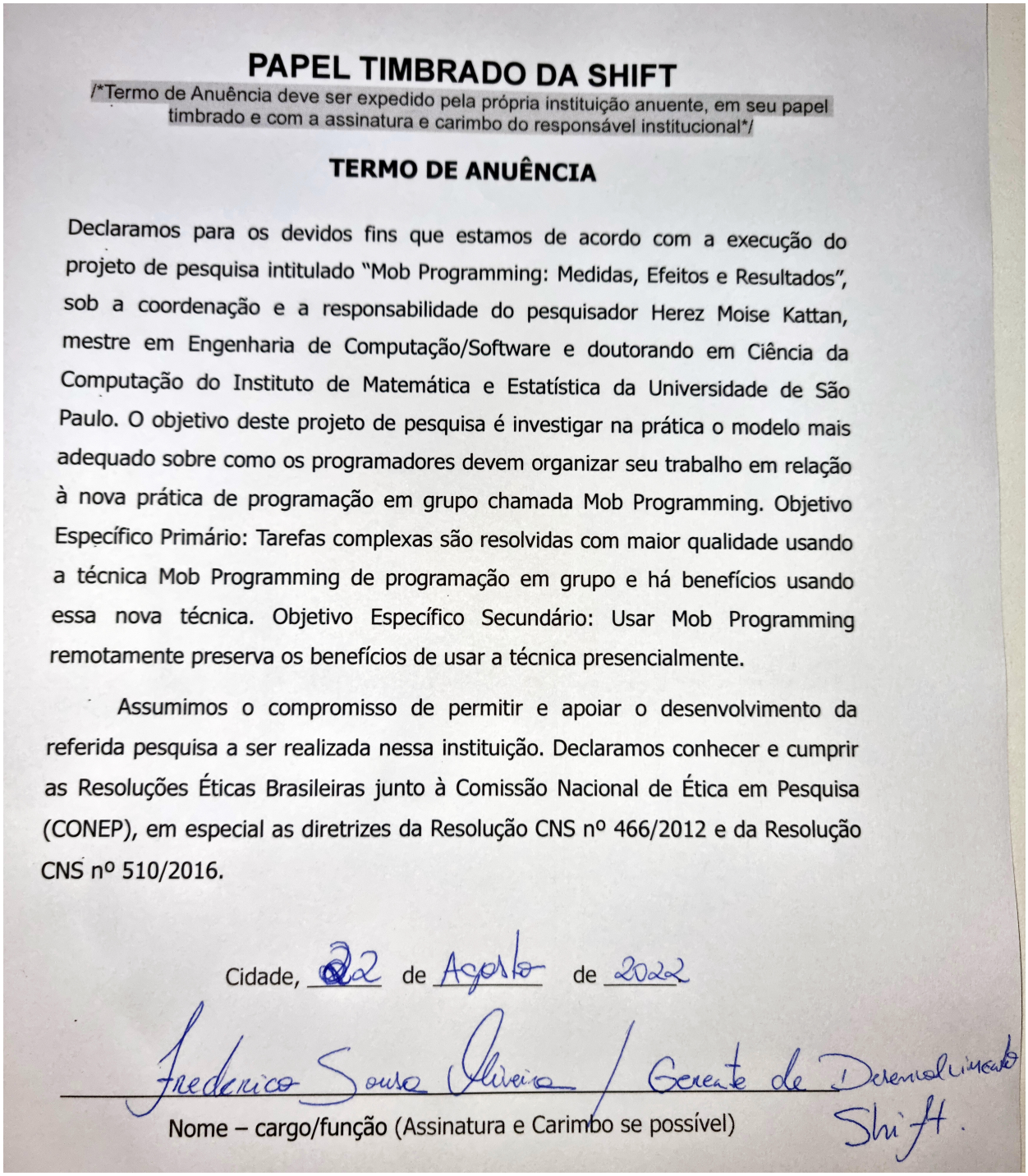
We are committed to allowing and supporting the development of this research, which considers our discussions related to agile and product development from a mob programming perspective. We have the right to review the text published about Mob Mentality Show before publication. We declare to know and comply with the Brazilian Ethical Resolutions with the National Commission of Ethics in Research (CONEP), including the guidelines of Brazilian National Council of Health Resolution Number 466/2012 and Brazilian National Council of Health Resolution Number 510/2016, respecting all voluntary participants rights.

San Marcos, 21 September 2022

Chris Lucian Austin Chadwick

Figure H.4: *Autorization of the Mob Mentality*

H.5 Written Approval of the Shift



PAPEL TIMBRADO DA SHIFT

/*Termo de Anuência deve ser expedido pela própria instituição anuente, em seu papel timbrado e com a assinatura e carimbo do responsável institucional*/

TERMO DE ANUÊNCIA

Declaramos para os devidos fins que estamos de acordo com a execução do projeto de pesquisa intitulado "Mob Programming: Medidas, Efeitos e Resultados", sob a coordenação e a responsabilidade do pesquisador Herez Moise Kattan, mestre em Engenharia de Computação/Software e doutorando em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. O objetivo deste projeto de pesquisa é investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada Mob Programming. Objetivo Específico Primário: Tarefas complexas são resolvidas com maior qualidade usando a técnica Mob Programming de programação em grupo e há benefícios usando essa nova técnica. Objetivo Específico Secundário: Usar Mob Programming remotamente preserva os benefícios de usar a técnica presencialmente.

Assumimos o compromisso de permitir e apoiar o desenvolvimento da referida pesquisa a ser realizada nessa instituição. Declaramos conhecer e cumprir as Resoluções Éticas Brasileiras junto à Comissão Nacional de Ética em Pesquisa (CONEP), em especial as diretrizes da Resolução CNS nº 466/2012 e da Resolução CNS nº 510/2016.

Cidade, 22 de Agosto de 2022

Frederico Sousa Oliveira / Gerente de Desenvolvimento Shift.

Nome – cargo/função (Assinatura e Carimbo se possível)

Figure H.5: *Autorization of the Shift*

H.6 Written Approval of the CoteiBem

síndiconet coteibem

TERMO DE ANUÊNCIA

Declaramos para os devidos fins que estamos de acordo com a execução do projeto de pesquisa intitulado "Mob Programming: Medidas, Efeitos e Resultados", sob a coordenação e a responsabilidade do pesquisador Herez Moise Kattan, mestre em Engenharia de Computação/Software e doutorando em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. O objetivo deste projeto de pesquisa é investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada Mob Programming. Objetivo Específico Primário: Tarefas complexas são resolvidas com maior qualidade usando a técnica Mob Programming de programação em grupo e há benefícios usando essa nova técnica. Objetivo Específico Secundário: Usar Mob Programming remotamente preserva os benefícios de usar a técnica presencialmente.

Assumimos o compromisso de permitir e apoiar o desenvolvimento da referida pesquisa a ser realizada nessa instituição. Declaramos conhecer e cumprir as Resoluções Éticas Brasileiras junto à Comissão Nacional de Ética em Pesquisa (CONEP), em especial as diretrizes da Resolução CNS nº 466/2012 e da Resolução CNS nº 510/2016.

São Paulo, Cidade, 09 de Setembro de 2022.

André Rocha Agostinho / CTO

SINDICONET TEC A. SIST. PUBL E PROPG LT

DocuSigned by: Andre Agostinho 4352E38B8D1F4B0...

Nome – cargo/função (Assinatura e Carimbo se possível)

Figure H.6: *Autorization of the CoteiBem*

H.7 Summary of Digital Approval of the CoteiBem

DocuSign

Certificado de Conclusão

Identificação de envelope: 8ACCD67C24CE4BEC88BE8A2BEA67DAEC — Status: Concluído

Assunto: Aqui está o seu documento assinado: Participacao_do_CoteiBem_em_doutorado_sobre_Mob_Programming_TERM

Envelope fonte:

Documentar páginas: 1	Assinaturas: 1	Remetente do envelope:
Certificar páginas: 1	Rubrica: 1	Andre Agostinho
Assinatura guiada: Desativado		andre@sindiconet.com.br
Selo com EnvelopeId (ID do envelope): Desativado		Endereço IP: 189.120.78.223
Fuso horário: (UTC-08:00) Hora do Pacífico (EUA e Canadá)		

Rastreamento de registros

Status: Original 09/09/2022 08:28:51	Portador: Andre Agostinho andre@sindiconet.com.br	Local: DocuSign

Eventos do signatário	Assinatura	Registro de hora e data
Andre Agostinho andre@sindiconet.com.br SINDICONET TEC A. SIST. PUBL E PROPG LT Nível de segurança: E-mail, Autenticação da conta (Nenhuma)	DocuSigned by: Andre Agostinho Adoção de assinatura: Estilo pré-selecionado Usando endereço IP: 189.120.78.223	Enviado: 09/09/2022 08:29:00 Visualizado: 09/09/2022 08:29:11 Assinado: 09/09/2022 08:36:54 Assinatura de forma livre

Termos de Assinatura e Registro Eletrônico:
Não disponível através da DocuSign

Eventos do signatário presencial	Assinatura	Registro de hora e data
Eventos de entrega do editor	**Status**	**Registro de hora e data**
Evento de entrega do agente	**Status**	**Registro de hora e data**
Eventos de entrega intermediários	**Status**	**Registro de hora e data**
Eventos de entrega certificados	**Status**	**Registro de hora e data**
Eventos de cópia	**Status**	**Registro de hora e data**
herez@ime.usp.br Nível de segurança: E-mail, Autenticação da conta (Nenhuma)	Copiado	Enviado: 09/09/2022 08:36:55 Visualizado: 09/09/2022 09:23:11

Termos de Assinatura e Registro Eletrônico:
Não disponível através da DocuSign

Eventos com testemunhas	Assinatura	Registro de hora e data
Eventos do tabelião	**Assinatura**	**Registro de hora e data**

Eventos de resumo do envelope	Status	Carimbo de data/hora
Envelope enviado	Com hash/criptografado	09/09/2022 08:29:00
Entrega certificada	Segurança verificada	09/09/2022 08:29:11
Assinatura concluída	Segurança verificada	09/09/2022 08:36:54
Concluído	Segurança verificada	09/09/2022 08:36:55
Eventos de pagamento	**Status**	**Carimbo de data/hora**

Figure H.7: *Summary of Digital Approval of the CoteiBem*

H.8 Written Approval of the SindicoNet

síndiconet

SINDICONET TEC A. SIST. PUBL E PROPG LTDA
02.015.852/0001-88

TERMO DE ANUÊNCIA

Declaramos para os devidos fins que estamos de acordo com a execução do projeto de pesquisa intitulado "Mob Programming: Medidas, Efeitos e Resultados", sob a coordenação e a responsabilidade do pesquisador Herez Moise Kattan, mestre em Engenharia de Computação/Software e doutorando em Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo. O objetivo deste projeto de pesquisa é investigar na prática o modelo mais adequado sobre como os programadores devem organizar seu trabalho em relação à nova prática de programação em grupo chamada Mob Programming. Objetivo Específico Primário: Tarefas complexas são resolvidas com maior qualidade usando a técnica Mob Programming de programação em grupo e há benefícios usando essa nova técnica. Objetivo Específico Secundário: Usar Mob Programming remotamente preserva os benefícios de usar a técnica presencialmente.

Assumimos o compromisso de permitir e apoiar o desenvolvimento da referida pesquisa a ser realizada nessa instituição. Declaramos conhecer e cumprir as Resoluções Éticas Brasileiras junto à Comissão Nacional de Ética em Pesquisa (CONEP), em especial as diretrizes da Resolução CNS nº 466/2012 e da Resolução CNS nº 510/2016.

São Paulo-SP, 09 de Setembro de 2022.

DocuSigned by:
Andre Agostinho
4352E3B88D1F4B0...

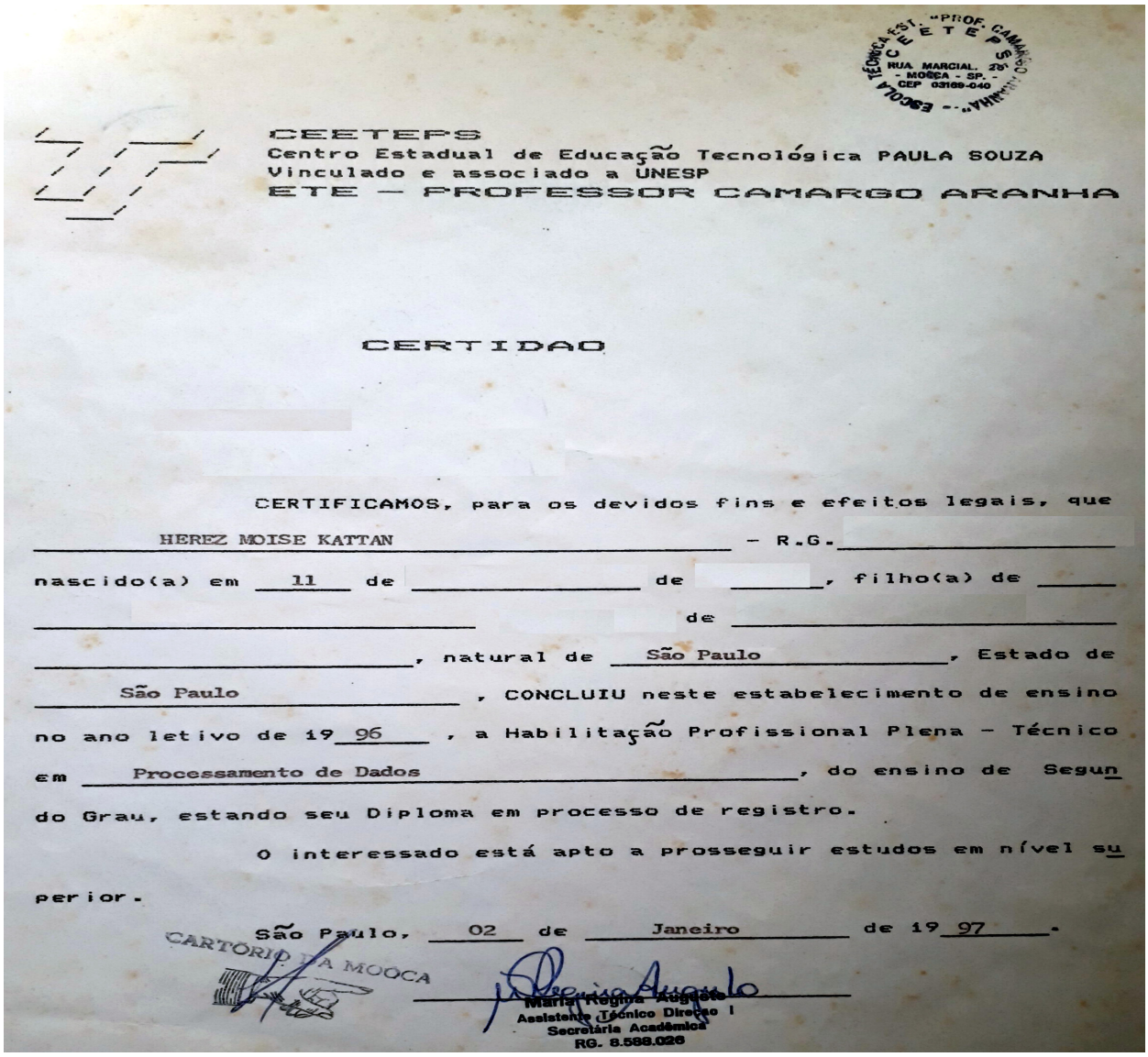
Andre Rocha Agostinho - Diretor de Tecnologia

Figure H.8: *Autorization of the SindicoNet*

Appendix I

Background of the Researcher

I.1 Paula Souza State Center for Technological Education



CEETEPS
Centro Estadual de Educação Tecnológica PAULA SOUZA
Vinculado e associado a UNESP
ETE – PROFESSOR CAMARGO ARANHA

CERTIDÃO

CERTIFICAMOS, para os devidos fins e efeitos legais, que HEREZ MOISE KATTAN – R.G. _____ nascido(a) em 11 de _____ de _____, filho(a) de _____ de _____, natural de São Paulo, Estado de São Paulo, CONCLUIU neste estabelecimento de ensino no ano letivo de 19 96, a Habilitação Profissional Plena – Técnico em Processamento de Dados, do ensino de Segundo Grau, estando seu Diploma em processo de registro.

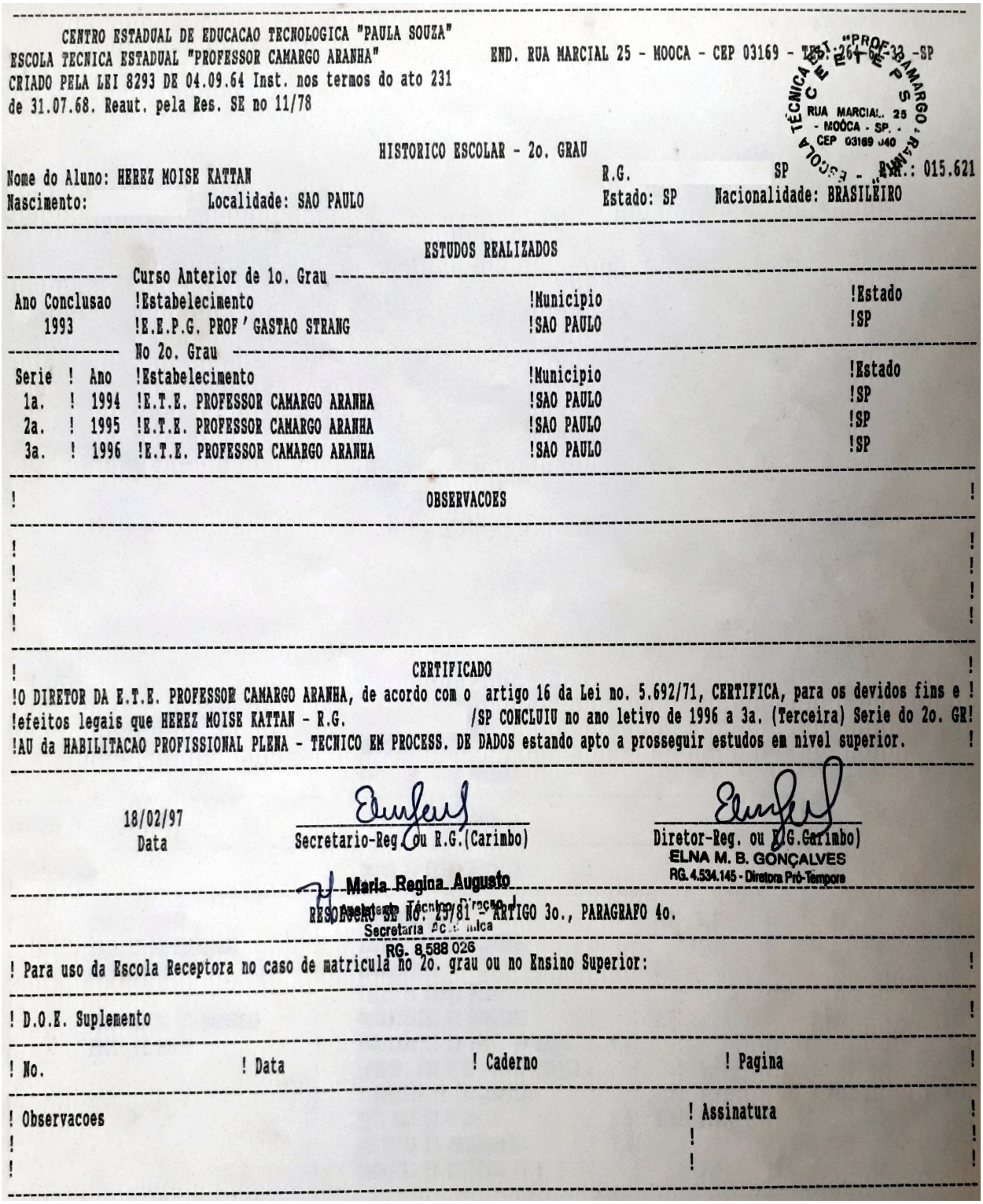
O interessado está apto a prosseguir estudos em nível superior.

São Paulo, 02 de Janeiro de 19 97.

CARTÓRIO DA MOOCA

Maria Regina Augusto
Assistente Técnico Diretor I
Secretaria Acadêmica
RG. 8.588.026

Figure I.1: *Full Professional Qualification in Data Processing Technician by the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*



CENTRO ESTADUAL DE EDUCACAO TECNOLOGICA "PAULA SOUZA"
ESCOLA TECNICA ESTADUAL "PROFESSOR CAMARGO ARANHA" END. RUA MARCIAL 25 - MOOCA - CEP 03169 - TEL. 261 6[illegible] -SP
CRIADO PELA LEI 8293 DE 04.09.64 Inst. nos termos do ato 231
de 31.07.68. Reaut. pela Res. SE no 11/78

ESCOLA TÉCNICA ESTADUAL "PROF. CAMARGO ARANHA" RUA MARCIAL, 25 - MOOCA - SP. - CEP 03169-040

HISTORICO ESCOLAR - 2o. GRAU

Nome do Aluno: HEREZ MOISE KATTAN R.G. SP - RM.: 015.621
Nascimento: Localidade: SAO PAULO Estado: SP Nacionalidade: BRASILEIRO

ESTUDOS REALIZADOS

Curso Anterior de 1o. Grau

Ano Conclusao	Estabelecimento	Municipio	Estado
1993	E.E.P.G. PROF'GASTAO STRANG	SAO PAULO	SP

No 2o. Grau

Serie	Ano	Estabelecimento	Municipio	Estado
1a.	1994	E.T.E. PROFESSOR CAMARGO ARANHA	SAO PAULO	SP
2a.	1995	E.T.E. PROFESSOR CAMARGO ARANHA	SAO PAULO	SP
3a.	1996	E.T.E. PROFESSOR CAMARGO ARANHA	SAO PAULO	SP

OBSERVACOES

CERTIFICADO

O DIRETOR DA E.T.E. PROFESSOR CAMARGO ARANHA, de acordo com o artigo 16 da Lei no. 5.692/71, CERTIFICA, para os devidos fins e efeitos legais que HEREZ MOISE KATTAN - R.G. /SP CONCLUIU no ano letivo de 1996 a 3a. (Terceira) Serie do 2o. GRAU da HABILITACAO PROFISSIONAL PLENA - TECNICO EM PROCESS. DE DADOS estando apto a prosseguir estudos em nivel superior.

18/02/97
Data

Secretario-Reg. ou R.G.(Carimbo)

Diretor-Reg. ou R.G.Carimbo)
ELNA M. B. GONÇALVES
RG. 4.534.145 - Diretora Pró-Tempore

Maria Regina Augusto
Assistente Técnico Diretoria
Secretaria Acadêmica
RG. 8.588 026

RESOLUCAO SE NO. 25/81 - ARTIGO 3o., PARAGRAFO 4o.

Para uso da Escola Receptora no caso de matricula no 2o. grau ou no Ensino Superior:

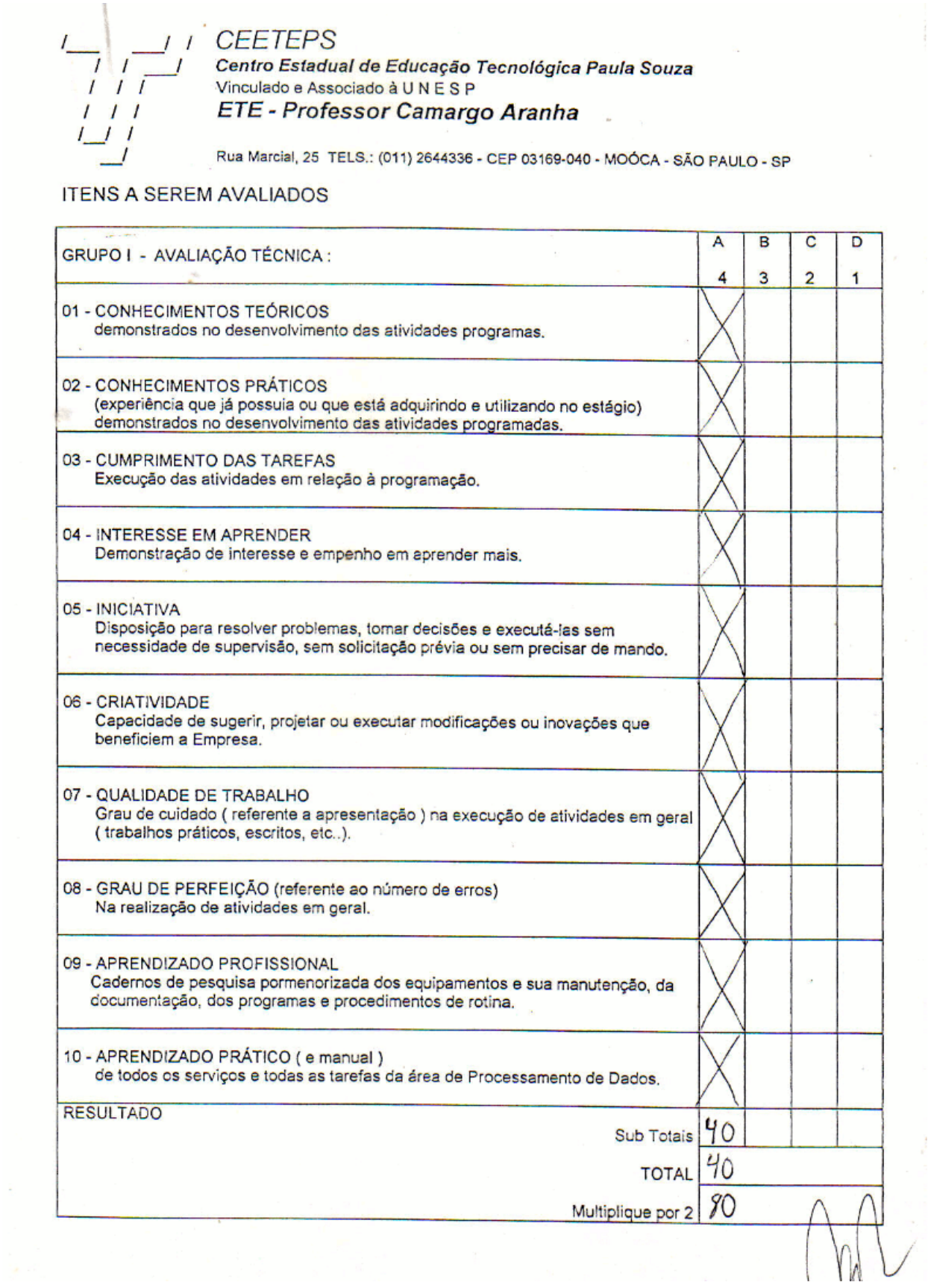
D.O.E. Suplemento

No.	Data	Caderno	Pagina

Observacoes

Assinatura

Figure I.2: *School Record of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*



CEETEPS
Centro Estadual de Educação Tecnológica Paula Souza
Vinculado e Associado à U N E S P
ETE - Professor Camargo Aranha
Rua Marcial, 25 TELS.: (011) 2644336 - CEP 03169-040 - MOÓCA - SÃO PAULO - SP

ITENS A SEREM AVALIADOS

GRUPO I - AVALIAÇÃO TÉCNICA :	A 4	B 3	C 2	D 1
01 - CONHECIMENTOS TEÓRICOS demonstrados no desenvolvimento das atividades programas.	X			
02 - CONHECIMENTOS PRÁTICOS (experiência que já possuia ou que está adquirindo e utilizando no estágio) demonstrados no desenvolvimento das atividades programadas.	X			
03 - CUMPRIMENTO DAS TAREFAS Execução das atividades em relação à programação.	X			
04 - INTERESSE EM APRENDER Demonstração de interesse e empenho em aprender mais.	X			
05 - INICIATIVA Disposição para resolver problemas, tomar decisões e executá-las sem necessidade de supervisão, sem solicitação prévia ou sem precisar de mando.	X			
06 - CRIATIVIDADE Capacidade de sugerir, projetar ou executar modificações ou inovações que beneficiem a Empresa.	X			
07 - QUALIDADE DE TRABALHO Grau de cuidado (referente a apresentação) na execução de atividades em geral (trabalhos práticos, escritos, etc..).	X			
08 - GRAU DE PERFEIÇÃO (referente ao número de erros) Na realização de atividades em geral.	X			
09 - APRENDIZADO PROFISSIONAL Cadernos de pesquisa pormenorizada dos equipamentos e sua manutenção, da documentação, dos programas e procedimentos de rotina.	X			
10 - APRENDIZADO PRÁTICO (e manual) de todos os serviços e todas as tarefas da área de Processamento de Dados.	X			
RESULTADO — Sub Totais	40			
TOTAL	40			
Multiplique por 2	80			

Figure I.3: *Page one of Internship Assessment of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*

CEETEPS
Centro Estadual de Educação Tecnológica Paula Souza
Vinculado e Associado à U N E S P
ETE - Professor Camargo Aranha
Rua Marcial, 25 TELS.: (011) 2644336 - CEP 03169-040 - MOÓCA - SÃO PAULO - SP

GRUPO I I - AVALIAÇÃO PESSOAL	A 4	B 3	C 2	D 1
01 - DISCIPLINA Observância das normas e dos regulamentos internos da empresa	X			
ASSIDUIDADE : 02 - Atividade, persistência	X			
03 - Presença e pontualidade	X			
04 - COOPERAÇÃO Disposição para cooperar com os colegas e atender nas atividades solicitadas	X			
05 - CONFIANÇA Discrição demonstrada quanto ao sigilo sobre as atividades confidenciais a ele confiadas (quando previamente avisado)	X			
06 - RESPONSABILIDADE Cuidado no uso das instalações, materiais, equipamentos, documentos ou qualquer outro bem de propriedade da empresa.	X			
07 - SOCIABILIDADE Facilidade de se integrar com os colegas e no ambiente de trabalho.	X			
EQUILÍBRIO EMOCIONAL : 08 - GRAU DE MATURIDADE e segurança no exercício de suas funções	X			
09 - GRAU DE AUTO-CONTROLE frente às situações problemáticas.	X			
10 - APRESENTAÇÃO PESSOAL Grau de higiene e cuidado com o vestuário	X			
RESULTADO DO GRUPO I I — Sub Totais	40			
TOTAL (b)	40			

TOTAL GERAL = SOMA DOS TOTAIS (a) + (b) = 80 + 40 = 120

NOTA DE APROVEITAMENTO (AVALIAÇÃO) = $\frac{\text{TOTAL GERAL}}{12} = \frac{120}{12} = 10$

CONCEITO :
- de 2,5 à 4,4 = Rejeitado ☐
- de 4,5 à 6,2 = Pode melhorar ☐
- de 6,3 à 8,0 = Bom ☐
- de 8,1 à 10,0 = Ótimo ☒

ASSINATURA DO AVALIADOR
SYNERSTAND Sistemas de Informação Ltda.

Figure I.4: *Page two of Internship Assessment of the Data Processing course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*

I.2 Institute for Technological Research



Instituto de Pesquisas Tecnológicas
do Estado de São Paulo

O Diretor Presidente do Instituto de Pesquisas Tecnológicas do Estado de São Paulo, no exercício de suas atribuições legais e tendo em vista a conclusão do Curso de Mestrado Profissional em Engenharia de Computação, Área de Concentração - Engenharia de Software, em 16 de dezembro de 2015, confere o título de

Mestre em Engenharia de Computação

a

Herez Moise Kattan

brasileiro, natural do Estado de São Paulo, nascido a de de 19 , RG n.º - SP,

e outorga-lhe o presente diploma
a fim de que possa gozar de todos os direitos e prerrogativas legais.

São Paulo, 15 de janeiro de 2016.

Prof. Dr. Eduardo Luiz Machado
Coordenador da CET

Prof. Dr. Fernando José Gomes Landgraf
Diretor Presidente

Diplomado

UNIVERSIDADE DE SÃO PAULO
SECRETARIA GERAL
DIVISÃO DE REGISTROS ACADÊMICOS
Diploma registrado sob nº 1708284
Processo nº 2016.1.110211.1.0
nos termos do Artigo 48 da Lei 9394/96.
São Paulo, 12 de maio de 2016
JOSE CARLOS GOMES
Técnico Acadêmico

De acordo.

Curso Reconhecido de acordo com o disposto na Portaria MEC n.º 1.077, de 31/08/2012, DOU de 13/09/2012.

Figure I.5: *Master of Science in Computer Engineering*

ipt INSTITUTO DE PESQUISA TECNOLÓGICA

Coordenadoria de Ensino Tecnológico - CET

MESTRADO PROFISSIONAL EM ENGENHARIA DE COMPUTAÇÃO

Curso reconhecido de acordo com o disposto na Portaria MEC nº 1.077 de 31/08/2012, publicada no Diário Oficial da União de 13/09/2012.

HISTÓRICO ESCOLAR

NOME: HEREZ MOISE KATTAN **MATRÍCULA Nº: 40800**

RG: (SP)

DATA DE NASCIMENTO:

LOCAL DE NASCIMENTO: Estado de São Paulo

NACIONALIDADE: Brasileira

DATA DE INGRESSO: 10/09/2012 **DATA DA DEFESA DA DISSERTAÇÃO:** 16/12/2015

ÁREA DE CONCENTRAÇÃO: Engenharia de Software

PROFICIÊNCIA EM LÍNGUA ESTRANGEIRA: Inglês - Aprovado

ORIENTADOR (a): Prof. Dr. José Eduardo Zindel Deboni

TÍTULO DA DISSERTAÇÃO: "Programação e revisão simultânea em pares: Uma extensão à programação em par".

CÓD	DISCIPLINAS	PERÍODO	CONCEITOS	CRÉDITOS	CARGA HORÁRIA
TOO–005	Orientação a Objetos: Fundamentos e UML	10/09/2012 21/12/2012	B	05	50
ESO - 001	Engenharia de Software	10/09/2012 21/12/2012	A	05	50
TIO-002	Tecnologias de Informação e Organizações	28/01/2013 10/05/2013	A	05	50
ERS-042	Engenharia de Requisitos de Software	28/01/2013 10/05/2013	A	05	50
PDS-052	Processos de Desenvolvimento de Software com Ênfase em UP Modelos	20/05/2013 30/08/2013	A	05	50

Av. Prof. Almeida Prado, 532 | Cidade Universitária
São Paulo | SP | CEP 05508-901
Tel 11 3767 4000 | Fax 11 3767 4002 | ipt@ipt.br
www.ipt.br

Figure I.6: *School Record from the Institute for Technological Research of the State of Sao Paulo (IPT) - page one.*

ipt INSTITUTO DE PESQUISA TECNOLÓGICA

NOME: HEREZ MOISE KATTAN **MATRÍCULA Nº: 40800**

CÓD	DISCIPLINAS	PERÍODO	CONCEITOS	CRÉDITOS	CARGA HORÁRIA
QGS-007	Qualidade e Melhoria do Processo de Geração de Software	20/05/2013 30/08/2013	A	05	50
TSO-021	Teste de Software	09/09/2013 20/12/2013	A	05	50
MPQ-036	Metodologia da Pesquisa	27/01/2014 09/05/2014	B	06	60
DIS-000	Exame de Qualificação	17/06/2015	AP	06	60
DIS-001	Defesa da Dissertação	16/12/2015	AP	06	60
TOTAL:				53	530

CONVENÇÕES: A= Excelente, B= Bom, C= Regular, D= Reprovado, I= Incompleto, AP= Aprovado
Conceitos: A, B, C e AP com direito a crédito.

CARGA HORÁRIA: 01 CRÉDITO = 10 horas

Nº MINIMO DE CRÉDITOS = 53
Disciplinas: 41
Exame de Qualificação: 06
Defesa de Dissertação: 06

Data da Expedição do Diploma: 15/01/2016

São Paulo, 15 de janeiro de 2016.

Coordenadoria de Ensino Tecnológico

Eduardo Luiz Machado
Coordenador da CET

Av. Prof. Almeida Prado, 532 | Cidade Universitária
São Paulo | SP | CEP 05508-901
Tel 11 3767 4000 | Fax 11 3767 4002 | ipt@ipt.br
www.ipt.br

Figure I.7: *School Record from the Institute for Technological Research of the State of Sao Paulo (IPT) - page two.*

I.3 Institute of Mathematics and Statistics

Universidade de São Paulo
Brasil

Janus
Sistema administrativo da Pós-graduação

Instituto de Matemática e Estatística

ATESTADO

Atestamos, para os devidos fins que, no período entre 01/01/2016 e 16/11/2022, o(a) senhor(a) Herez Moise Kattan, de número USP 9860455, cursou a(s) disciplina(s) abaixo na qualidade de aluno(a) regular do programa de pós-graduação em Ciência da Computação.

Disciplina: Linguagens, Autômatos e Computabilidade
Sigla: MAC4722-2/2 **Carga Horária:** 120 **Conceito:** B **Frequência:** 100 **Créditos:** 8
Início: 22/02/2016 **Término:** 10/06/2016

Disciplina: Programação Orientada a Objetos
Sigla: MAC5714-5/4 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8
Início: 22/02/2016 **Término:** 10/06/2016

Disciplina: Laboratório de Programação Extrema
Sigla: MAC5716-4/2 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8
Início: 08/08/2016 **Término:** 25/11/2016

Disciplina: Inteligência Artificial em Jogos de Computador
Sigla: MAC5784-2/2 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8
Início: 08/08/2016 **Término:** 25/11/2016

Documento emitido às 00:01:02 horas do dia 16/11/2022 (hora e data de Brasília).
Código de controle: IP7I - LZYP - RY1F - 417N
Código de controle válido até: 16/11/2023 Página 1 de 3

Figure I.8: *School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page one.*

Universidade de São Paulo
Brasil

Janus
Sistema administrativo da Pós-graduação

Instituto de Matemática e Estatística

Aluno: Herez Moise Kattan, número USP 9860455

Programa: Ciência da Computação.

Disciplina: Metodologia de Pesquisa para Ciência da Computação

Sigla: MAC6910-1/2 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8

Início: 13/03/2017 **Término:** 30/06/2017

Disciplina: Lean Startup

Sigla: MAC6924-1/1 **Carga Horária:** 30 **Conceito:** A **Frequência:** 100 **Créditos:** 2

Início: 06/03/2017 **Término:** 20/03/2017

Disciplina: Tópicos Especiais em Computação e Biodiversidade

Sigla: PCS5026-2/1 **Carga Horária:** 30 **Conceito:** A **Frequência:** 100 **Créditos:** 2

Início: 27/04/2017 **Término:** 03/05/2017

Disciplina: Tópicos Avançados em Engenharia de Computação

Sigla: PCS5119-1/1 **Carga Horária:** 30 **Conceito:** A **Frequência:** 100 **Créditos:** 2

Início: 19/02/2018 **Término:** 04/03/2018

Disciplina: Métodos Qualitativos de Pesquisa

Sigla: MAC6932-1/1 **Carga Horária:** 30 **Conceito:** A **Frequência:** 100 **Créditos:** 2

Início: 21/03/2018 **Término:** 27/03/2018

Disciplina: Laboratório Avançado de Métodos Ágeis de Desenvolvimento de Software

Sigla: MAC5717-3/2 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8

Início: 06/08/2018 **Término:** 23/11/2018

Documento emitido às 00:01:02 horas do dia 16/11/2022 (hora e data de Brasília).

Código de controle: IP7I - LZYP - RY1F - 417N

Código de controle válido até: 16/11/2023 Página 2 de 3

Figure I.9: *School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page two.*

Universidade de São Paulo
Brasil

Janus
Sistema administrativo da Pós-graduação

Instituto de Matemática e Estatística

Aluno: Herez Moise Kattan, número USP 9860455

Programa: Ciência da Computação.

Disciplina: Estudos Avançados em Sistemas Sócio-Técnicos

Sigla: MAC6933-1/1 **Carga Horária:** 120 **Conceito:** A **Frequência:** 100 **Créditos:** 8

Início: 06/08/2018 **Término:** 23/11/2018

Disciplina: Escrita Acadêmica em Inglês

Sigla: IME4001-1/1 **Carga Horária:** 60 **Conceito:** B **Frequência:** 100 **Créditos:** 4

Início: 02/10/2019 **Término:** 13/11/2019

Conceito a partir de 02/01/1997:
A - Excelente, com direito a crédito; B - Bom, com direito a crédito; C - Regular, com direito a crédito; R - Reprovado; T - Transferência.
Um(1) crédito equivale a 15 horas de atividade programada.

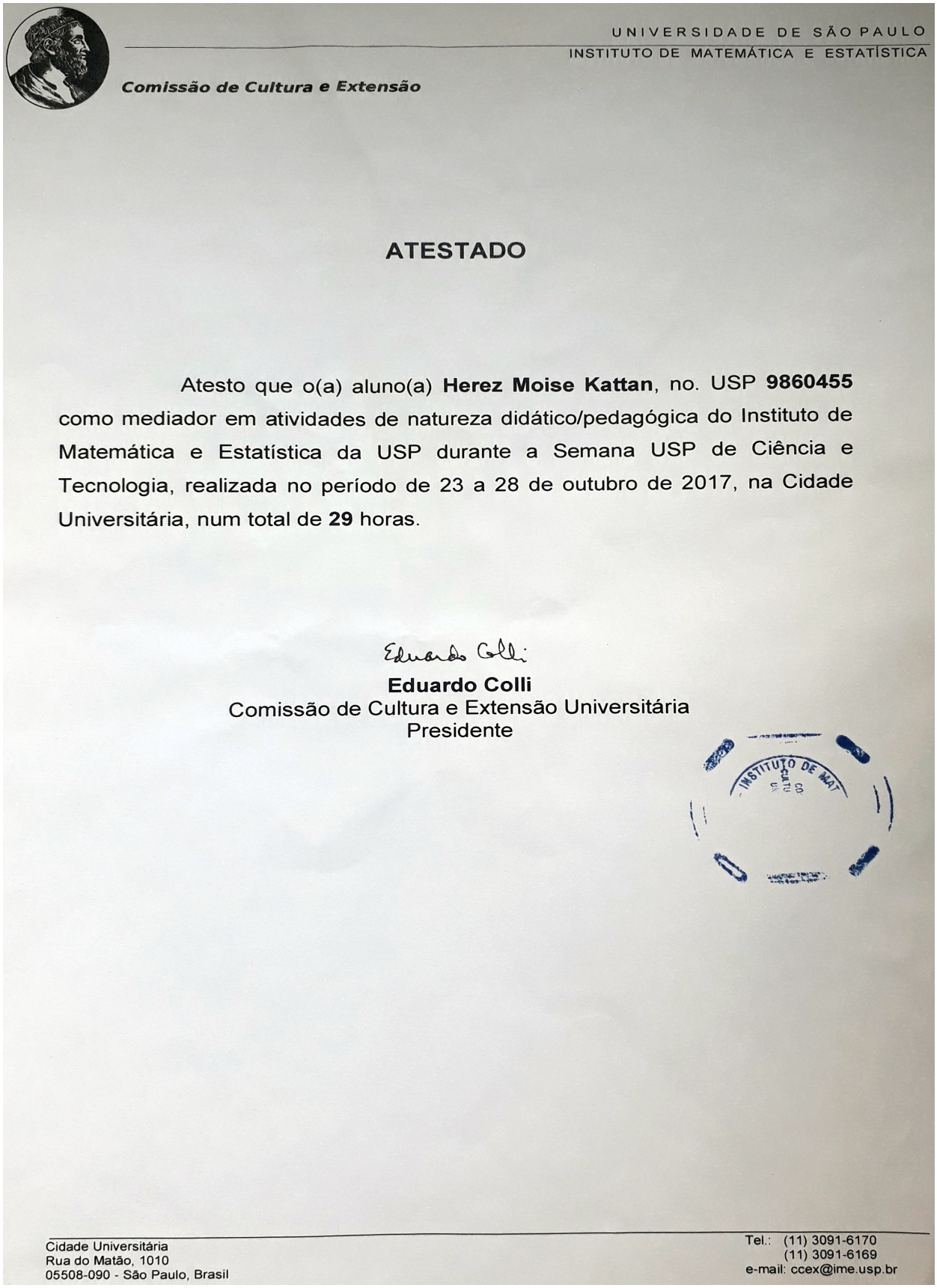
Este documento eletrônico dispensa carimbo e assinatura. Sua autenticidade pode ser comprovada fornecendo-se o código de controle na seguinte página da Universidade de São Paulo: **https://uspdigital.usp.br/iddigital**

Documento emitido às 00:01:02 horas do dia 16/11/2022 (hora e data de Brasília).

Código de controle: IP7I - LZYP - RY1F - 417N

Código de controle válido até: 16/11/2023 Página 3 de 3

Figure I.10: *School Record from the Institute of Mathematics and Statistics of the University of Sao Paulo (IME-USP) - page three.*



UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA

Comissão de Cultura e Extensão

ATESTADO

Atesto que o(a) aluno(a) **Herez Moise Kattan**, no. USP **9860455** como mediador em atividades de natureza didático/pedagógica do Instituto de Matemática e Estatística da USP durante a Semana USP de Ciência e Tecnologia, realizada no período de 23 a 28 de outubro de 2017, na Cidade Universitária, num total de **29** horas.

Eduardo Colli

Eduardo Colli
Comissão de Cultura e Extensão Universitária
Presidente

Cidade Universitária
Rua do Matão, 1010
05508-090 - São Paulo, Brasil

Tel.: (11) 3091-6170
(11) 3091-6169
e-mail: ccex@ime.usp.br

Figure I.11: *Certificate working as a moderator in the Science and Technology Week from the University of Sao Paulo.*

Alfredo Goldman
USP-CCSL FLOSS Competence Center - Director
Professor/Researcher at IME-USP
gold@ime.usp.br

November 2016, São Paulo - Brazil

Recommendation Letter

Herez Moise Kattan received the Technical degree in data processing from ETECA (State Technical School Camargo Aranha), São Paulo, in 1996, the Bachelor degree in analysis of systems from UNIP (Paulista University), São Paulo, in 2000, the Master degree in computing/software engineering from IPT (Technological Research Institute of São Paulo State), São Paulo City, in 2015 and now is a Doctoral candidate in science of computing at IME-USP (Mathematics and Statistics Institute of São Paulo University), Brazil.

It's my absolute pleasure to recommend Herez Moise Kattan for collaborate with CEM. I was member of master's thesis committee that assessment him. After attesting his solid contribution I invited him to do the PhD under my supervision.

I thoroughly enjoyed my time working with Herez, and came to know him as a truly valuable asset to absolutely our team. He is honest, dependable and hard-working. His knowledge of software development and expertise in software development methods was a huge advantage to our entire team.

Along with his undeniable talent, Herez has always been an absolute joy to work with. He is a true team player, and always manages to foster positive discussions and bring the best out of other students.

Without a doubt, I confidently recommend Herez to join your team at CEM. As a dedicated and knowledgeable worker and an all-around great person, I know that he will be a beneficial addition to your organization.

Please feel free to contact me at gold@ime.usp.br should you like to discuss Herez's qualifications and experience further. I'd be more than happy to expand on my recommendation if needed.

Sincerely,

Alfredo Goldman
Centro de Competência
em Software Livre
IME - USP

Figure I.12: *Recommendation letter from FLOSS Competence Center of USP.*

I.4 Certificates of Courses


SOFT LINE Cursos e Programação

C.G.C. 71.523.674/0001-19

Certificado

Certifico que Reuez Moise Kattan

Concluiu o curso de Nivel 01 Ministrado Nesta Instituição de Ensino, foi aprovado nos exames finais, fazendo jús ao presente Certificado.

São Paulo, 13 de Dezembro de 1993

Direção

Coordenação

Figure I.13: *Certificate of Course in Programming, in an early stage of my career as a teenager. Also, I put another from my childhood regarding the programming course card in subsection 3.6 "Role of the Researcher".*



CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA
"PAULA SOUZA"
Vinculado e Associado a UNESP
ETE de São Paulo

CERTIFICADO

Certificamos que **Herez Moise Kattan** , participou do Curso "ISO 9000", promovido pela Escola Técnica Estadual de São Paulo, com apoio da ABNT (Associação Brasileira de Normas Técnicas) e Escola Técnica Estadual Professor Camargo Aranha, no período de 05 a 07 de Abril de 1995.

Carga Horária: 16 horas

São Paulo, 07 de Abril de 1995.

PAULO SÉRGIO PALAZZO ROLIM
Docente

LAURA M.J.LAGANÁ DIETZOLD
Diretora da ETE de São Paulo

TEREZINHA DE JESUS T. DE SOUZA
Diretora da ETE Professor Camargo Aranha

Figure I.14: *Certificate of the ISO 9000 course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*


Associação de Pais e Mestres
Escola Técnica Estadual
de São Paulo

CERTIFICADO

Certificamos que, **Herez Moise Kattan** participou do Curso Visual Basic promovido pela Escola Técnica Estadual de São Paulo, no período de 24 a 28 de Julho 1995.

Carga Horária: 20 horas

São Paulo, 28 de Julho de 1995.

Reinaldo Madarazo

REINALDO MADARAZO
Docente

LAURA M. J. LAGANÁ DIETZOLD
Pres. do Cons. Delib. da APM da ETE São Paulo

Figure I.15: *Certificate of the Visual Basic course from the State Center for Technological Education Paula Souza linked to Universidade Estadual Paulista (UNESP).*



Figure I.16: *Certificate of the Visual Interdev course from the Microsoft authorized technical education center G&P linked to Microsoft Official Curriculum.*



Figure I.17: *Certificate of the Windows NT Core Technologies course from the Microsoft authorized technical education center G&P linked to Microsoft Official Curriculum.*



Figure I.18: *Certification of the Managing the Classroom Environment from the Microsoft authorized technical education center Impacta linked to Microsoft Official Curriculum.*



Figure I.19: *Certificate of the FIX Protocol course from the Sao Paulo Stock Exchange and the Brazilian Mercantile Futures Exchange, now called B3, is linked to CME Group.*

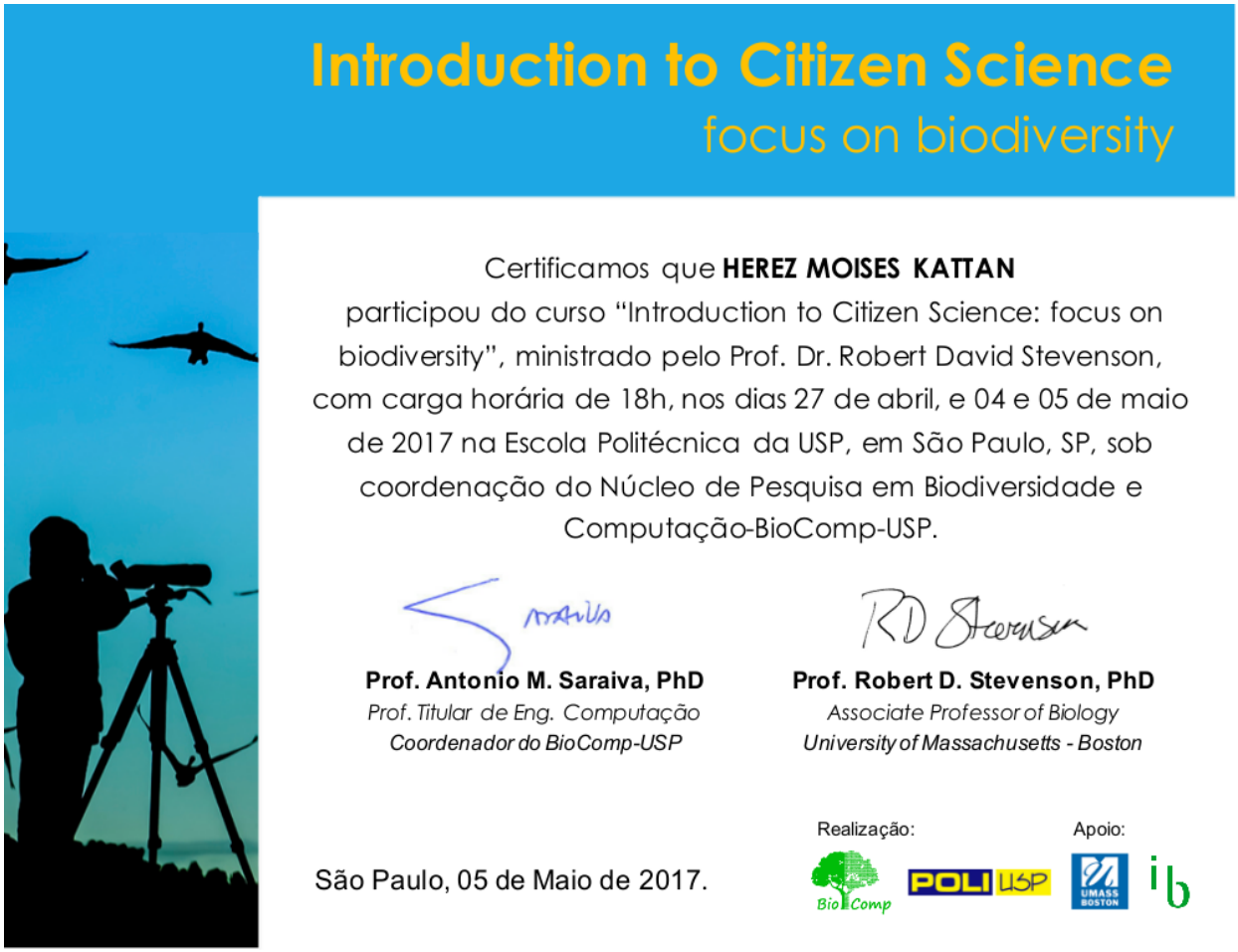
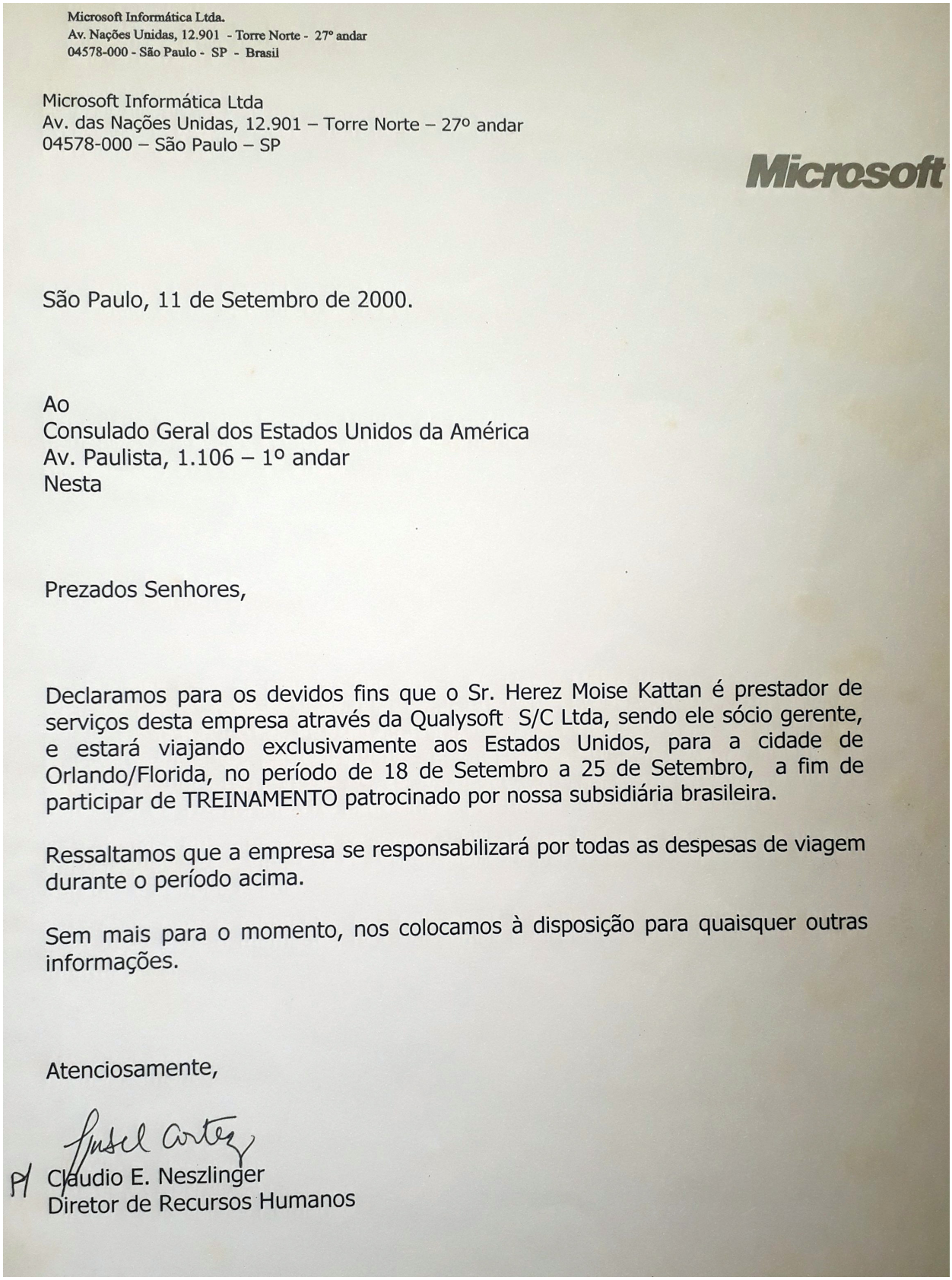


Figure I.20: *Certificate of the Introduction to Citizen Science focus on biodiversity course from USP and University of Massachusetts.*



Figure I.21: *Certificate of the Network course.*

I.5 Microsoft



Microsoft Informática Ltda.
Av. Nações Unidas, 12.901 - Torre Norte - 27º andar
04578-000 - São Paulo - SP - Brasil

Microsoft Informática Ltda
Av. das Nações Unidas, 12.901 – Torre Norte – 27º andar
04578-000 – São Paulo – SP

Microsoft

São Paulo, 11 de Setembro de 2000.

Ao
Consulado Geral dos Estados Unidos da América
Av. Paulista, 1.106 – 1º andar
Nesta

Prezados Senhores,

Declaramos para os devidos fins que o Sr. Herez Moise Kattan é prestador de serviços desta empresa através da Qualysoft S/C Ltda, sendo ele sócio gerente, e estará viajando exclusivamente aos Estados Unidos, para a cidade de Orlando/Florida, no período de 18 de Setembro a 25 de Setembro, a fim de participar de TREINAMENTO patrocinado por nossa subsidiária brasileira.

Ressaltamos que a empresa se responsabilizará por todas as despesas de viagem durante o período acima.

Sem mais para o momento, nos colocamos à disposição para quaisquer outras informações.

Atenciosamente,

pl Claudio E. Neszlinger
Diretor de Recursos Humanos

Figure I.22: *Letter from Microsoft to the United States of America Consulate General in Brazil for the researcher author of this thesis to have a Visa.*

Microsoft®

Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0
Examination Score Report

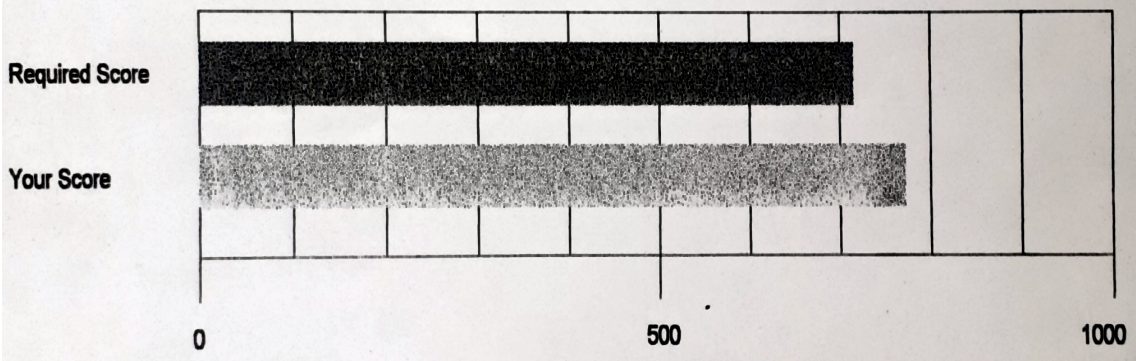
CANDIDATE: Herez Kattan

CANDIDATE ID: 1780147 **DATE:** 02-Mar-2000

REGISTRATION NUMBER: 853838 **SITE NUMBER:** 40267

EXAM: Designing and Implementing Desktop Applications with Microsoft® Visual Basic® 6.0

SERIES: 070-176



PASSING SCORE: 714 YOUR SCORE: 771 GRADE: Pass

Section Analysis	Percent Correct
Deriving the Physical Design	77%
Establishing the Development Environment	100%
Creating User Services	70%
Creating and Managing COM Components	61%
Creating Data Services	66%
Testing the Solution	100%
Deploying an Application	77%
Maintaining and Supporting an Application	100%

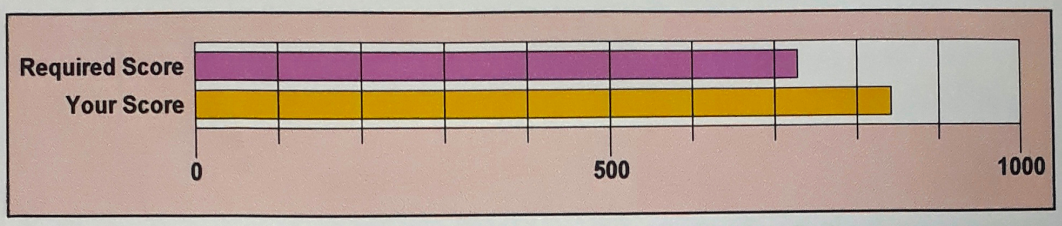
Figure I.23: *Examination Score Report Microsoft' Designing and Implementing Desktop Applications.*

Microsoft®

Designing and Implementing Distributed Applications with Microsoft® Visual Basic® 6.0
Examination Score Report

CANDIDATE: HEREZ MOISE KATTAN
CANDIDATE ID: SP2464797 **DATE:** April 29, 2004
REGISTRATION NUMBER: K6FLAM50F7 **SITE NUMBER:** BR31

EXAM: Designing and Implementing Distributed Applications with Microsoft® Visual Basic® 6.0
SERIES: 175



PASSING SCORE: 728 YOUR SCORE: 842 GRADE: Pass

Section Analysis	Percent Correct
1. Developing the Conceptual and Logical Design	100%
2. Deriving the Physical Design	88%
3. Establishing the Development Environment	66%
4. Creating User Services	81%
5. Creating and Managing COM Components	92%
6. Creating Data Services	81%
7. Testing the Solution	85%
8. Deploying an Application	60%
9. Maintaining and Supporting an Application	100%

DO NOT LOSE THIS REPORT

This examination was delivered at an Authorized Prometric Testing Center. To register for another Microsoft exam in the United States or Canada, call 1-800-755-EXAM. Elsewhere, please contact your Sylvan Prometric regional service center.

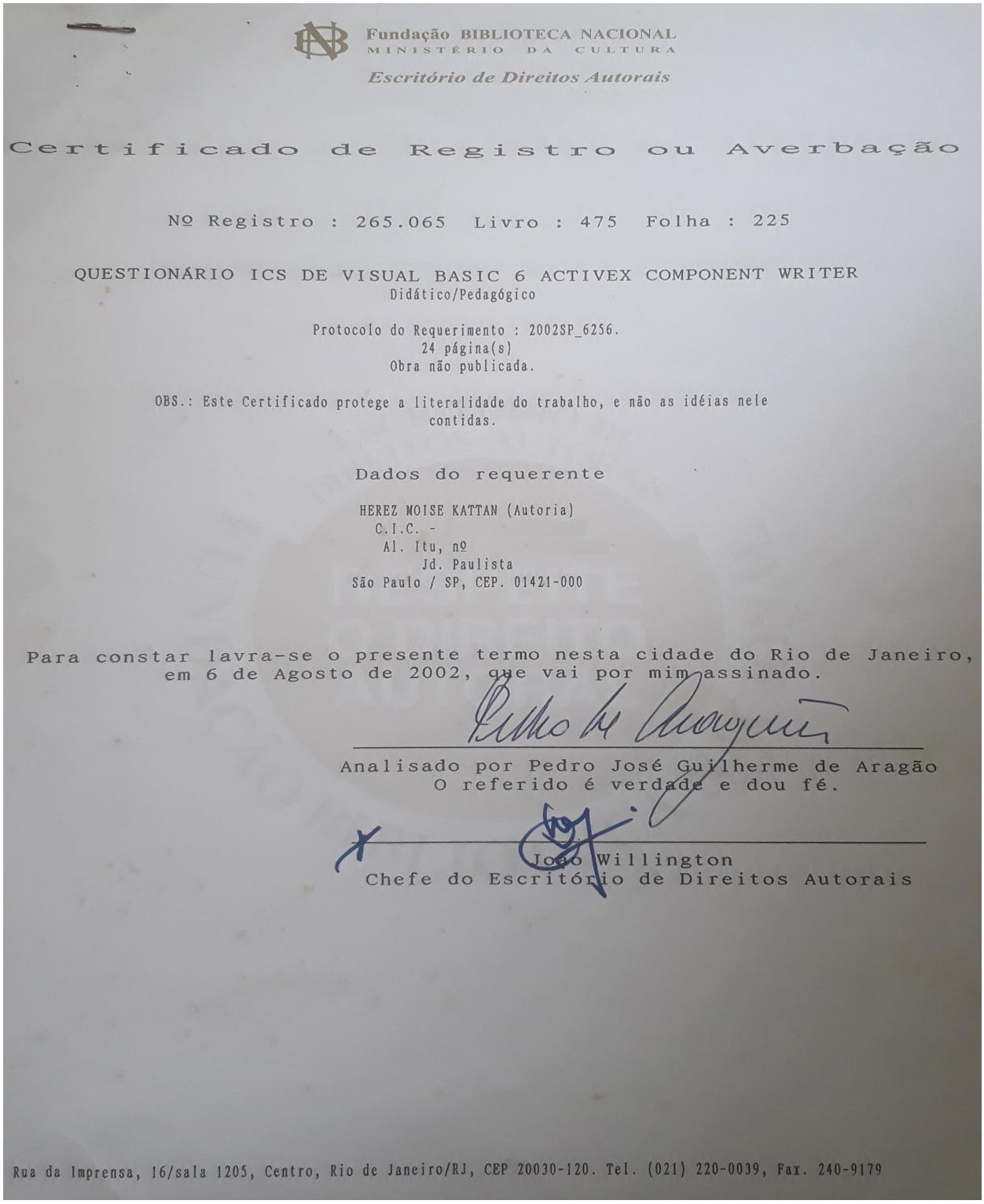
Figure I.24: *Examination Score Report Microsoft' Designing and Implementing Distributed Applications.*

I.6 Delphi



Figure I.25: *Recommendation letter from Delphi.*

I.7 Author



Fundação BIBLIOTECA NACIONAL
MINISTÉRIO DA CULTURA
Escritório de Direitos Autorais

Certificado de Registro ou Averbação

Nº Registro : 265.065 Livro : 475 Folha : 225

QUESTIONÁRIO ICS DE VISUAL BASIC 6 ACTIVEX COMPONENT WRITER
Didático/Pedagógico

Protocolo do Requerimento : 2002SP_6256.
24 página(s)
Obra não publicada.

OBS.: Este Certificado protege a literalidade do trabalho, e não as idéias nele contidas.

Dados do requerente

HEREZ MOISE KATTAN (Autoria)
C.I.C. -
Al. Itu, nº
Jd. Paulista
São Paulo / SP, CEP. 01421-000

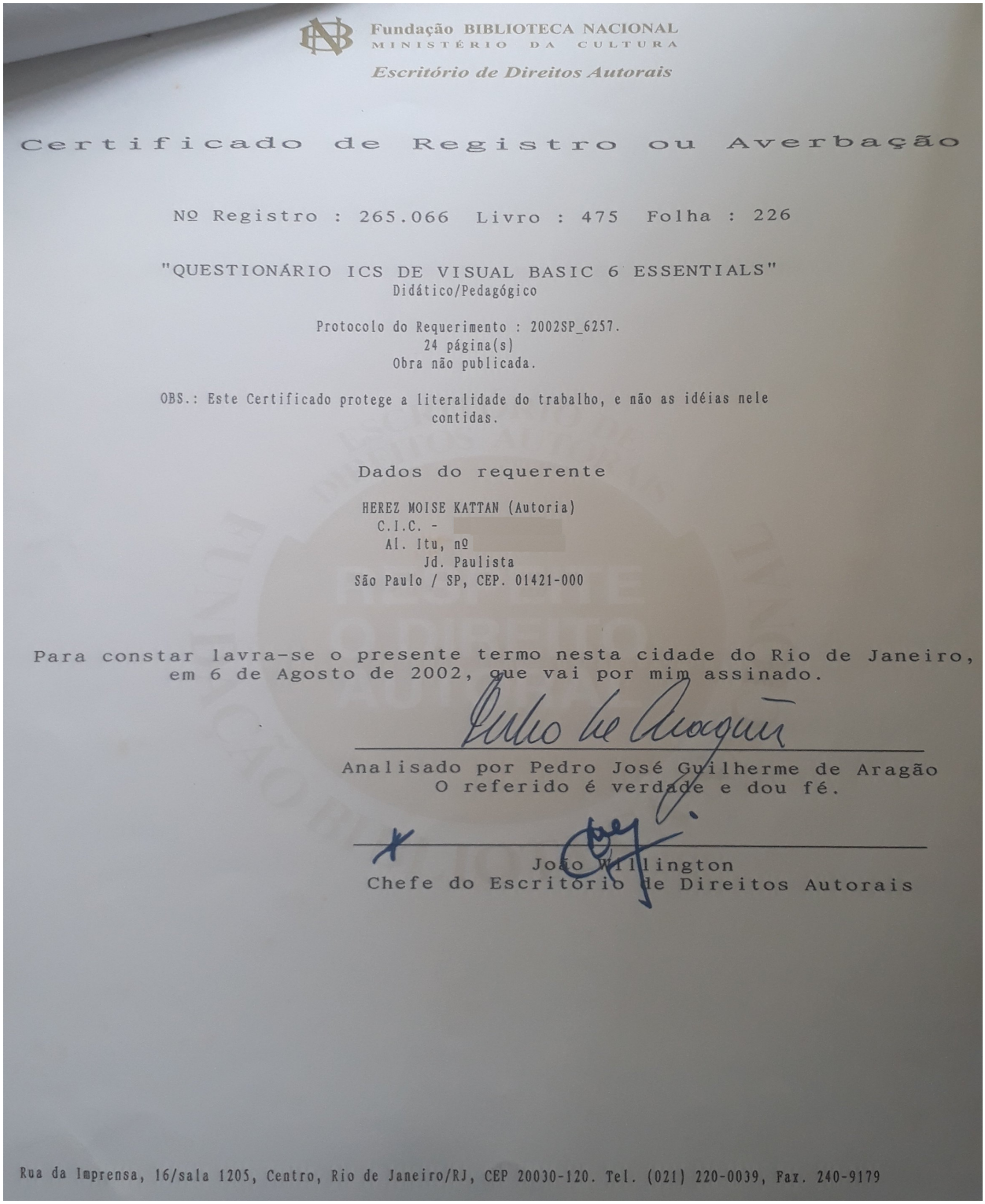
Para constar lavra-se o presente termo nesta cidade do Rio de Janeiro, em 6 de Agosto de 2002, que vai por mim assinado.

Analisado por Pedro José Guilherme de Aragão
O referido é verdade e dou fé.

João Willington
Chefe do Escritório de Direitos Autorais

Rua da Imprensa, 16/sala 1205, Centro, Rio de Janeiro/RJ, CEP 20030-120. Tel. (021) 220-0039, Fax. 240-9179

Figure I.26: *Register Number 265.065 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998.*



Fundação BIBLIOTECA NACIONAL
MINISTÉRIO DA CULTURA
Escritório de Direitos Autorais

Certificado de Registro ou Averbação

Nº Registro : 265.066 Livro : 475 Folha : 226

"QUESTIONÁRIO ICS DE VISUAL BASIC 6 ESSENTIALS"
Didático/Pedagógico

Protocolo do Requerimento : 2002SP_6257.
24 página(s)
Obra não publicada.

OBS.: Este Certificado protege a literalidade do trabalho, e não as idéias nele contidas.

Dados do requerente

HEREZ MOISE KATTAN (Autoria)
C.I.C. -
Al. Itu, nº
Jd. Paulista
São Paulo / SP, CEP. 01421-000

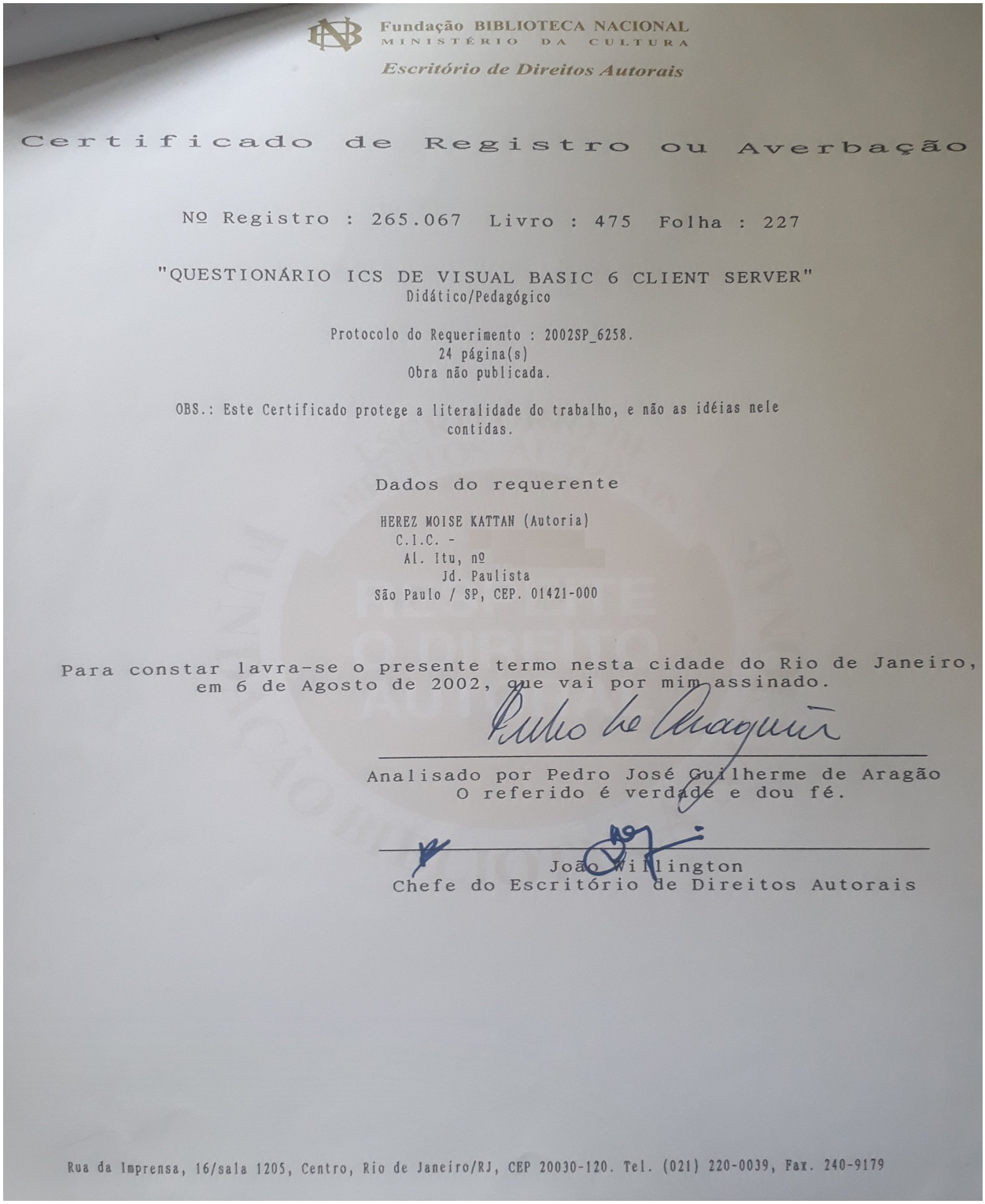
Para constar lavra-se o presente termo nesta cidade do Rio de Janeiro, em 6 de Agosto de 2002, que vai por mim assinado.

Analisado por Pedro José Guilherme de Aragão
O referido é verdade e dou fé.

João Willington
Chefe do Escritório de Direitos Autorais

Rua da Imprensa, 16/sala 1205, Centro, Rio de Janeiro/RJ, CEP 20030-120. Tel. (021) 220-0039, Fax. 240-9179

Figure I.27: *Register Number 265.066 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998.*



Fundação BIBLIOTECA NACIONAL
MINISTÉRIO DA CULTURA
Escritório de Direitos Autorais

Certificado de Registro ou Averbação

Nº Registro : 265.067 Livro : 475 Folha : 227

"QUESTIONÁRIO ICS DE VISUAL BASIC 6 CLIENT SERVER"
Didático/Pedagógico

Protocolo do Requerimento : 2002SP_6258.
24 página(s)
Obra não publicada.

OBS.: Este Certificado protege a literalidade do trabalho, e não as idéias nele contidas.

Dados do requerente

HEREZ MOISE KATTAN (Autoria)
C.I.C. -
Al. Itu, nº
Jd. Paulista
São Paulo / SP, CEP. 01421-000

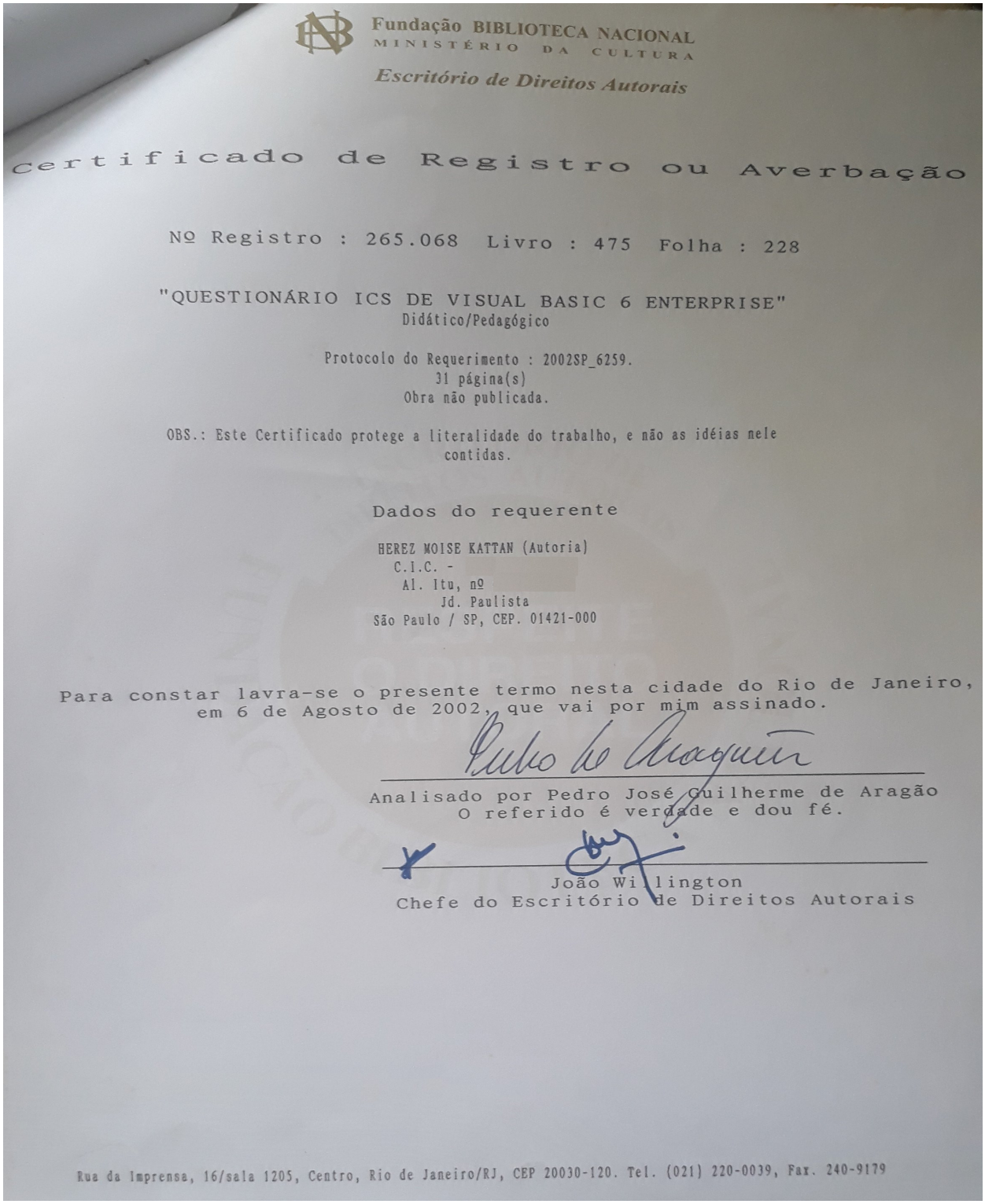
Para constar lavra-se o presente termo nesta cidade do Rio de Janeiro, em 6 de Agosto de 2002, que vai por mim assinado.

Analisado por Pedro José Guilherme de Aragão
O referido é verdade e dou fé.

João Willington
Chefe do Escritório de Direitos Autorais

Rua da Imprensa, 16/sala 1205, Centro, Rio de Janeiro/RJ, CEP 20030-120. Tel. (021) 220-0039, Fax. 240-9179

Figure I.28: *Register Number 265.067 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998.*



Fundação BIBLIOTECA NACIONAL
MINISTÉRIO DA CULTURA
Escritório de Direitos Autorais

Certificado de Registro ou Averbação

Nº Registro : 265.068 Livro : 475 Folha : 228

"QUESTIONÁRIO ICS DE VISUAL BASIC 6 ENTERPRISE"
Didático/Pedagógico

Protocolo do Requerimento : 2002SP_6259.
31 página(s)
Obra não publicada.

OBS.: Este Certificado protege a literalidade do trabalho, e não as idéias nele contidas.

Dados do requerente

HEREZ MOISE KATTAN (Autoria)
C.I.C. -
Al. Itu, nº
Jd. Paulista
São Paulo / SP, CEP. 01421-000

Para constar lavra-se o presente termo nesta cidade do Rio de Janeiro, em 6 de Agosto de 2002, que vai por mim assinado.

Analisado por Pedro José Guilherme de Aragão
O referido é verdade e dou fé.

João Willington
Chefe do Escritório de Direitos Autorais

Rua da Imprensa, 16/sala 1205, Centro, Rio de Janeiro/RJ, CEP 20030-120. Tel. (021) 220-0039, Fax. 240-9179

Figure I.29: *Register Number 265.068 of my material in the National Library, which has been responsible for registering intellectual works since 1898, when the first specific Brazilian law on copyright was published until today, through the Copyright Office, it offers this service and other related services to Brazilian citizens, for the legal security of moral and patrimonial rights of the author, under the terms of law n. 9.610, of February 19, 1998.*

I.8 TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research



Figure I.30: *TK95 Color Computer - Original Box - Front View.*



Figure I.31: *TK95 Color Computer - Original Box - Lateral View. Note the circle, compatilhe with ZX Spectrum.*

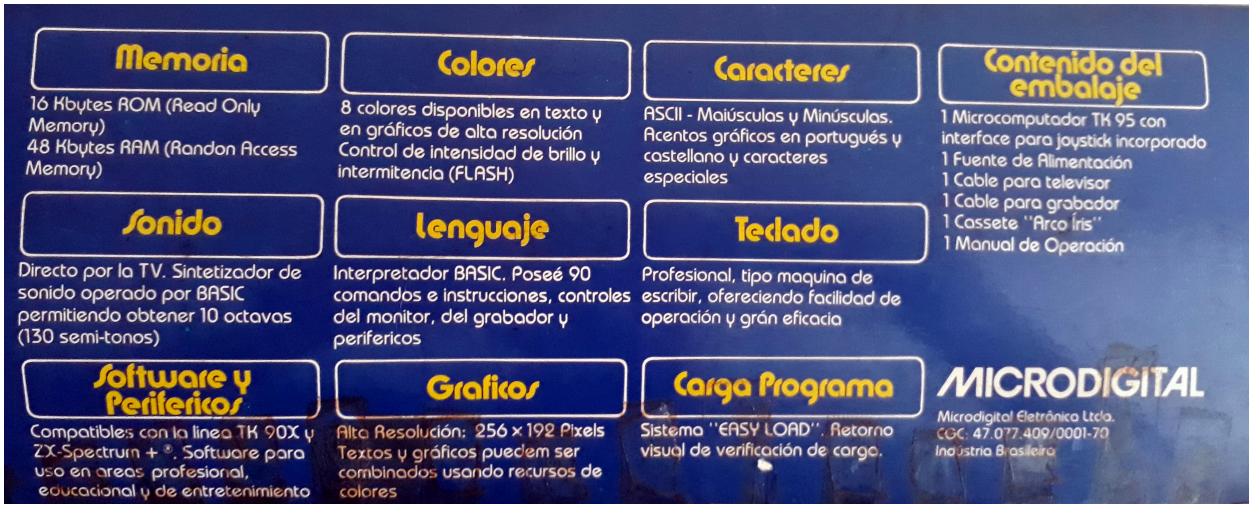


Figure I.32: *TK95 Color Computer - Original Box - Lateral View. Note that one side is written in Portuguese, but the other is written in Spanish because Brazil exports to Latin America.*

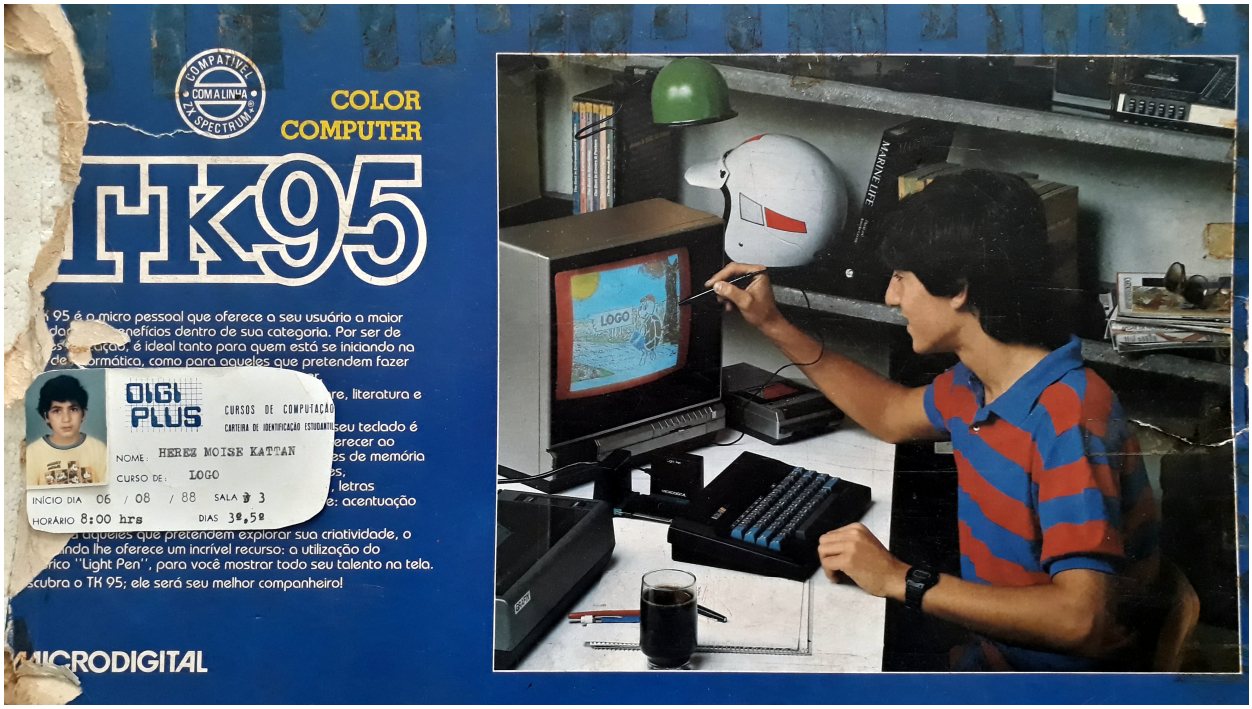


Figure I.33: *TK95 Color Computer - Original Box - Back View. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the box original of TK 95, a ZX Spectrum Brazilian Clone, working on TV, with three peripherals, the recorder to save the data in cassette tapes, the matrix printer, and the lighting pen to draw on the screen.*

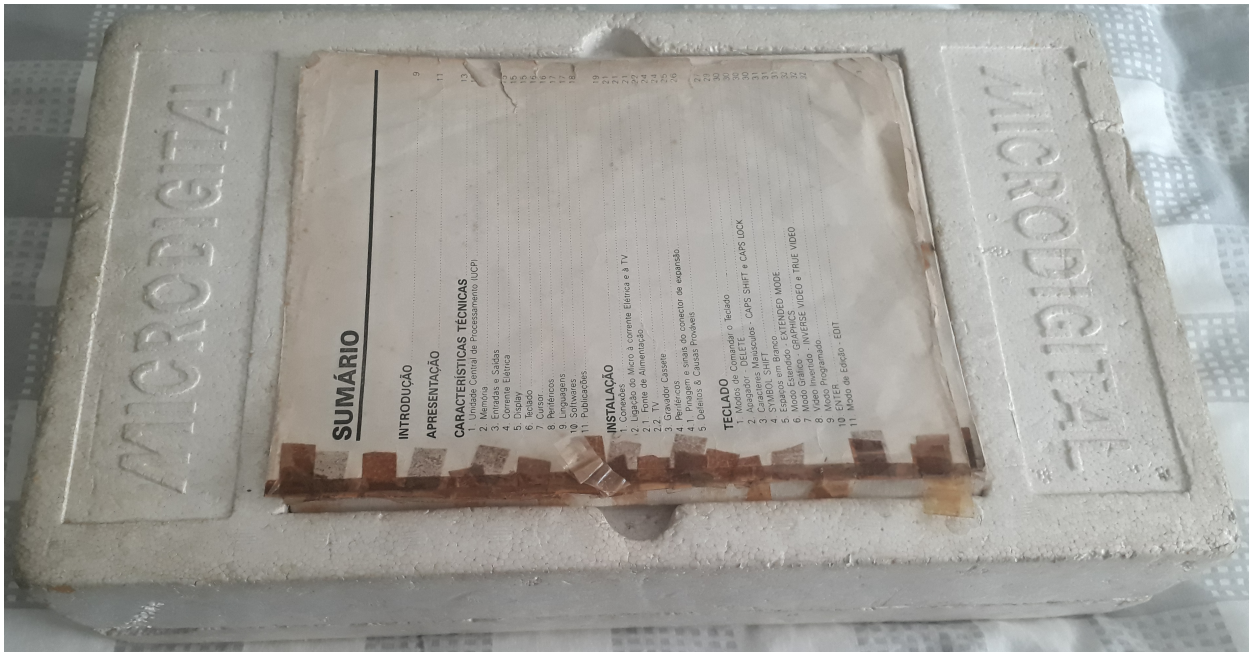


Figure I.34: *Original styrofoam box, it helped preserve the computer for 40 years. But the paper of the original instructions book has suffered over time.*



Figure I.35: *Software stored in the cassette tape. They have two sides. The data are flawlessly stored after 40 years.*



Figure I.36: *Lado B of the cassette tape. They have two sides. The data are flawlessly stored after 40 years.*



Figure I.37: *TK95 Color Computer inside the Original Box. Note a cassette tape on the left superior corner. It was the method to persist your data. On the one hand, it is limited. On the other hand, forty years later, the data are stored with no one sector damaged, and the software runs on the computer without compatibility issues. Moreover, it is years of light ahead of modern computers in energy saving, which is crucial nowadays because of climate change. This is a treasure to be studied.*



Figure I.38: *TK95 Color Computer Keyboard. Note the BASIC Language commands are in the keys.*



Figure I.39: *Front view of the AC power adapter of the TK95 Color Computer.*



Figure I.40: *TK95 Color Computer - Back View. The first connector on the right is to plug the TV. The others in the sequence are to connect the earphones, tape recorder, and microphone. Following are the connectors for the joystick and the next one for peripherals, e. g., the lighting pen shown in the back view of the box. The last is to connect the AC power adapter.*



Figure I.41: *TK95, the Brazilian clone of the ZX Spectrum from Sinclair Research in the original box. Note that I put my computer course card with my photo from when a child. Moreover, the photo shows the some books Isaaman et al. (1982), ISaaman and Tyler (1982), Waters et al. (1983).*

I.9 Diploma of Honor to Merit from the city of Sao Paulo



Figure I.42: *Diploma of Honor to Merit from the city of Sao Paulo signed by the epoch Mayor Mario Covas*[1]*, who after became the majority leader of the Constitutional Assembly from Brazil, which drew up a new democratic charter.*

1. Mario Covas, politician, born April 21, 1930; died March 6, 2001. Once the military relinquished power, Covas became the majority leader of the Constitutional Assembly, which drew up a new democratic charter. The constitution was probably Covas's most lasting contribution to Brazil. In 1986, Covas was elected to the senate with 7.8m votes. One of the seven founders of the Brazilian Social Democratic Party in 1988, Mario Covas was its candidate for president in the first direct elections since the end of the 1964-1985 dictatorship. Fernando Henrique Cardoso was re-elected in 1998 from his Party, and Covas was the obvious choice for 2002 until he was diagnosed with bladder cancer shortly after Cardoso's second victory. In his career as a politician, he was elected federal representative, mayor of Sao Paulo City (1983-1985), senator, and twice Governor of SÃ£o Paulo (1994-1998 and 1998-2001). He was a founder and member of PMDB (Party of the Brazilian Democratic Movement) and later PSDB (Brazilian Social Democracy Party).

I.10 The First Letter of Saint Paul to the Corinthians



Figure I.43: *Author: Mary Harrsch, Title: St Paul outside St Peter's Basilica in Vatican City. Photographed in St. Peter's Basilica in Vatican City. Available at: Flickr.com in her profile MHarrsch in the album The Vatican under the Creative Commons license: Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0).*

1 If I speak in the tongues[1] of men or of angels,
but do not have love, I am only a resounding
gong or a clanging cymbal.
2 If I have the gift of prophecy and can fathom
all mysteries and all knowledge, and if I have a
faith that can move mountains, but do not have
love, I am nothing.
3 If I give all I possess to the poor and give over
my body to hardship that I may boast,[2] but do
not have love, I gain nothing.
4 Love is patient, love is kind. It does not envy,
it does not boast, it is not proud.
5 It does not dishonor others, it is not self-
seeking, it is not easily angered, it keeps no
record of wrongs.
6 Love does not delight in evil but rejoices with
the truth.
7 It always protects, always trusts, always
hopes, always perseveres.
8 Love never fails. But where there are prophe-
cies, they will cease; where there are tongues,
they will be stilled; where there is knowledge, it
will pass away.
9 For we know in part and we prophesy in part,
10 but when completeness comes, what is in part
disappears.
11 When I was a child, I talked like a child,
I thought like a child, I reasoned like a child.
When I became a man, I put the ways of child-
hood behind me.
12 For now we see only a reflection as in a mirror; then we shall see face to face. Now I know in
part; then I shall know fully, even as I am fully known.
13 And now these three remain: faith, hope and love. But the greatest of these is love.

Figure I.43 is St Paul outside St Peter's Basilica in Vatican City. Photographed in St. Peter's Basilica in Vatican City by Mary Harrsch, who kindly put it on Flickr, it permits collaborations all over the world, available at flickr.com/photos/mharrsch/6776460573. This loving act allowance this Appendix to get more beautiful. The power of Love corroborates the achievement of an innovative and safer Communication Space. Thus, readily and pleasantly more relevant tasks could be done with less wasting and unneeded effort.

According to Britannica the First Letter of Paul to the Corinthians, probably written about 53-54 CE at Ephesus, Asia Minor, deals with problems that arose in the early years after Paul's initial missionary visit (c. 50-51) to Corinth.

Corinthians 13:1 tongues[1] Or *languages*
Corinthians 13:3 boast[2] Some manuscripts *body to the flames*

[1] Corinthians 13:1 Or languages
[2] Corinthians 13:3 Some manuscripts body to the flames

Bibliography

Adamson *et al.***(2014)** David Adamson, Gregory Dyke, Hyeju Jang and Carolyn Penstein Rosé. Towards an agile approach to adapting dynamic collaboration support to student needs. *International Journal of Artificial Intelligence in Education*, 24(1):92–124. ISSN 1560-4306. doi: 10.1007/s40593-013-0012-6. URL https://doi.org/10.1007/s40593-013-0012-6. Cited on pag. 44

Adeliyi *et al.***(2021)** Adeola Adeliyi, Janet Hughes, Karen Kear, Bobby Law, Brendan Murphy, Jon Rosewell, Ann Walshe and Michel Wermelinger. Remote pair programming. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, page 1289, New York, NY, USA. Association for Computing Machinery. ISBN 9781450380621. doi: 10.1145/3408877.3439681. URL https://doi.org/10.1145/3408877.3439681. Cited on pag. 29

Adolph *et al.***(2008)** Steve Adolph, Wendy Hall and Philippe Kruchten. A methodological leg to stand on: lessons learned using grounded theory to study software development. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, page 13. ACM. Cited on pag. 90, 170

Al-Jarrah and Pontelli(2016) Ahmad Al-Jarrah and Enrico Pontelli. *On the Effectiveness of a Collaborative Virtual Pair-Programming Environment*, pages 583–595. Springer International Publishing, Cham. ISBN 978-3-319-39483-1. doi: 10.1007/978-3-319-39483-1_53. URL https://doi.org/10.1007/978-3-319-39483-1_53. Cited on pag. 25

Allan(2006) George Allan. The legitimacy of grounded theory. In *Proceedings of the 5th European Conference on Research Methods in Business and Management*, ECRM 2006, pages 1–8. Academic Conferences Ltd (July 17, 2006). ISBN 978-1905305261. Cited on pag. 71, 91

Allan(2003) George Allan. A critique of using grounded theory as a research method. *Electronic journal of business research methods*, 2(1):pp1–10. Cited on pag. 86

Allen Parrish and Hale(2004) David Hale Allen Parrish, Randy Smith and Joanne Hale. A field study of developer pairs: productivity impacts and implications. *IEEE Software*, 21(5):76 – 79. ISSN 0740-7459. doi: 10.1109/MS.2004.1331306. Cited on pag. 2, 24

Analizo(2018) Analizo. Analizo static code metrics collector. WebSite of Analizo, an independent part of Mezuro Project, 2018. URL http://www.analizo.org/. Cited on pag. 6

Ancona and Caldwell(1988) Deborah Gladstein Ancona and David F. Caldwell. Beyond task and maintenance: Defining external functions in groups. *Group & Organization Studies*, 13(4): 468–494. doi: 10.1177/105960118801300405. URL https://doi.org/10.1177/105960118801300405. Cited on pag. 107, 108

Anderfelt(2020) Victor Anderfelt. From mob programming to mob development : User-centred design in collaborative software development. Master Thesis, Uppsala University, Department of Informatics and Media. Cited on pag. 41

Anderson and McMillan(2003) Carl Anderson and Elizabeth McMillan. Of ants and men: self organization in human and insect teams. *Emergence: Complexity and Organization*, 5(2):29–41. URL http://oro.open.ac.uk/2629/. Cited on pag. 108, 172

Antinyan *et al.***(2015)** V. Antinyan, M. Staron, J. Derehag, M. Runsten, E. WikstrÃ¶m, W. Meding, A. Henriksson and J. Hansson. Identifying complex functions: By investigating various aspects of code complexity. In *2015 Science and Information Conference (SAI)*, pages 879–888. doi: 10.1109/SAI.2015.7237246. Cited on pag. 52

Arisholm *et al.***(2007)** Erik Arisholm, Hans Gallis, Tore Dyba and Dag I.K. Sjoberg. Evaluating pair programming with respect to system complexity and programmer expertise. *IEEE Trans. Soft. Eng.*, 33(2):65–86. ISSN 0098-5589. doi: 10.1109/TSE.2007.17. URL http://dx.doi.org/10.1109/TSE.2007.17. Cited on pag. 2, 26

Aune *et al.***(2018)** Ole Kristian Aune, Christian Echtermeyer and Elias Sørensen. Mob programming: A qualitative study from the perspective of a development team. *Research Gate*. Cited on pag. 53

Balijepally *et al.***(2017)** Venugopal Balijepally, Sumera Chaudhry and Sridhar P. Nerur. Mob programming - A promising innovation in the agile toolkit. In *23rd Americas Conference on Information Systems, AMCIS 2017, Boston, MA, USA, August 10-12, 2017*. URL http://aisel.aisnet.org/amcis2017/SystemsAnalysis/Presentations/10. Cited on pag. 15, 39

Bandukda and Nasir(2010) M. Bandukda and Z. Nasir. Efficacy of distributed pair programming. In *2010 International Conference on Information and Emerging Technologies*, pages 1–6. doi: 10.1109/ICIET.2010.5625667. Cited on pag. 28

Beck(1999) Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77. ISSN 0018-9162. doi: 10.1109/2.796139. URL http://dx.doi.org/10.1109/2.796139. Cited on pag. 24

Beck(2000) Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-61641-6. Cited on pag. xvii, 25, 141

Beck and Andres(2004) Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional. ISBN 0321278658. Cited on pag. 30, 61

Begel and Nagappan(2008) Andrew Begel and Nachi Nagappan. Pair programming: What's in it for me? In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 120 – 128, New York, NY, USA. ACM. ISBN 978-1-59593-971-5. doi: 10.1145/1414004.1414026. URL http://doi.acm.org/10.1145/1414004.1414026. Cited on pag. 1, 24, 25

Belbin(2012) R Meredith Belbin. *Team roles at work*. Routledge. Cited on pag. 107

Benoliel(1996) Jeanne Quint Benoliel. Grounded theory and nursing knowledge. *Qualitative Health Research*, 6(3):406–428. doi: 10.1177/104973239600600308. URL https://doi.org/10.1177/104973239600600308. Cited on pag. 60

Björklund and Fridebo(2020) Philip Björklund and Jacob Fridebo. Mob programming and its impact on the developer's well-being and individual performance, 2020. Cited on pag. 52

Björklund *et al.***(2020)** Philip Björklund, Jacob Fridebo and Fisnik Dalipi. Exploring the impact of mob programming on the well-being of the developers : Insights from a software company. In *Proceedings of the NOKOBIT conference 24-25th of November 2020 :*, volume 28 of *NOKOBIT - Norsk konferanse for organisasjoners bruk av informasjonsteknologi*. URL https://ojs.bibsys.no/index.php/Nokobit/article/view/862. Cited on pag. 41

Boehm *et al.***(2000)** Barry W. Boehm, Ellis Horowitz, Ray Madachy, Donald Reifer, Bradford K. Clark, Bert Steece, Winsor A. Brown, Sunita Chulani and Chris Abts. *Software Cost Estimation with Cocomo II (with CD-ROM)*. Prentice Hall PTR, New Jersey. ISBN 0130266922. URL http://www.amazon.fr/exec/obidos/ASIN/0130266922/citeulike04-21. Cited on pag. 53

Boekhout(2016) Karel Boekhout. *Mob Programming: Find Fun Faster*, pages 185–192. Springer International Publishing, Cham. ISBN 978-3-319-33515-5. doi: 10.1007/978-3-319-33515-5_15. URL https://doi.org/10.1007/978-3-319-33515-5_15. Cited on pag. 39, 51

Bosu and Carver(2013) A. Bosu and J. C. Carver. Impact of peer code review on peer impression formation: A survey. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 133–142. doi: 10.1109/ESEM.2013.23. Cited on pag. 78, 80, 81

Brasil(1980) Constituição Brasil. Lei nº 6.880, de 9 de dezembro de 1980: Dispõe sobre o estatuto dos militares. *Diário Oficial da União*. Cited on pag. 10

Bravo and Goldman(2010) Mariana Bravo and Alfredo Goldman. *Reinforcing the Learning of Agile Practices Using Coding Dojos*, pages 379–380. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-13054-0. doi: 10.1007/978-3-642-13054-0_41. URL https://doi.org/10.1007/978-3-642-13054-0_41. Cited on pag. 37

Britannica() The Editors of Encyclopaedia. Britannica. Letters of paul to the corinthians. https://www.britannica.com/topic/The-Letter-of-Paul-to-the-Corinthians. Accessed: 2022-12-9. Cited on pag. 177, 255

Brooker(2009) Will Brooker. Maps of many worlds: Remembering computer game fandom in the 1980s. *Transformative Works and Cultures*, 2. doi: 10.3983/twc.2009.034. URL https://doi.org/10.3983/twc.2009.034. Cited on pag. 66

Brooks(1975) Fred P. Brooks, Jr. The mythical man-month. *SIGPLAN Not.*, 10(6):193–. ISSN 0362-1340. doi: 10.1145/390016.808439. URL http://doi.acm.org/10.1145/390016.808439. Cited on pag. 24, 25, 30

Brown(2010) Brené Brown. *The gifts of imperfection: Let go of who you think you're supposed to be and embrace who you are.* Simon and Schuster. Cited on pag. 139, 140

Brown(2015) Brené Brown. *Daring greatly: How the courage to be vulnerable transforms the way we live, love, parent, and lead.* Penguin. Cited on pag. 139

Bryant and Charmaz(2010) A. Bryant and K. Charmaz. *The SAGE Handbook of Grounded Theory: Paperback Edition.* SAGE Publications. ISBN 9781446203743. URL https://books.google.com.br/books?id=OrgZjp9CoN8C. Cited on pag. 72

Bryant and Charmaz(2007) Antony Bryant and Kathy Charmaz. *The Sage handbook of grounded theory.* Sage. Cited on pag. 90

Buchan and Pearl(2018) Jim Buchan and Mark Pearl. Leveraging the mob mentality: An experience report on mob programming. In *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018*, EASE'18, pages 199–204, New York, NY, USA. ACM. ISBN 978-1-4503-6403-4. doi: 10.1145/3210459.3210482. URL http://doi.acm.org/10.1145/3210459.3210482. Cited on pag. 40

Budgen *et al.***(2006)** David Budgen, Stuart Charters, Mark Turner, Pearl Brereton, Barbara Kitchenham and Stephen Linkman. Investigating the applicability of the evidence-based paradigm to software engineering. In *Proceedings of the 2006 International Workshop on Workshop on Interdisciplinary Software Engineering Research*, WISER, pages 7–14, New York, NY, USA. ACM. ISBN 1-59593-409-X. doi: 10.1145/1137661.1137665. URL http://doi.acm.org/10.1145/1137661.1137665. Cited on pag. 59, 61

Busechian *et al.*(2018) Sara Busechian, Vladimir Ivanov, Alan Rogers, Ilyas Sirazitdinov, Giancarlo Succi, Alexander Tormasov and Jooyong Yi. Understanding the impact of pair programming on the minds of developers. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '18, pages 85–88, New York, NY, USA. ACM. ISBN 978-1-4503-5662-6. doi: 10.1145/3183399.3183413. URL http://doi.acm.org/10.1145/3183399.3183413. Cited on pag. 33, 44

Carmines and Zeller(1979) E.G. Carmines and R.A. Zeller. *Reliability and Validity Assessment*. Quantitative Applications in the Social Sciences. SAGE Publications. ISBN 9781452207711. URL https://books.google.com.br/books?id=o5x1AwAAQBAJ. Cited on pag. 59

Chamberlain-Salaun *et al.*(2013) Jennifer Chamberlain-Salaun, Jane Mills and Kim Usher. Linking symbolic interactionism and grounded theory methods in a research design: From corbin and strauss assumptions to action. *SAGE Open*, 3(3):2158244013505757. doi: 10.1177/2158244013505757. URL https://doi.org/10.1177/2158244013505757. Cited on pag. 73

Chametzky(2016) Barry Chametzky. Coding in classic grounded theory: I've done an interview; now what? *Sociol. mind*, 06(04):163–172. Cited on pag. 85, 86

Charmaz(2008) K Charmaz. Grounded theory as an emergent method. in, sn hesse-biber, & p. leavy (eds.), handbook of emergent method (pp. 155-170), 2008. Cited on pag. xix, 74

Charmaz(2014) K. Charmaz. *Constructing Grounded Theory*. Introducing Qualitative Methods series. SAGE Publications. ISBN 9780857029140. URL https://books.google.com.br/books?id=k7R_NAEACAAJ. Cited on pag. 73, 74, 75, 84, 90

Charmaz(1990) Kathy Charmaz. Discovering chronic illness: Using grounded theory. *Social Science and Medicine*, 30(11):1161 – 1172. ISSN 0277-9536. doi: https://doi.org/10.1016/0277-9536(90)90256-R. URL http://www.sciencedirect.com/science/article/pii/027795369090256R. Special Issue Qualitative Research On Chronic Illness. Cited on pag. 60

Charmaz(2006) Kathy Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. sage. Cited on pag. 74

Charmaz(2016) Kathy Charmaz. Shifting the grounds: Constructivist grounded theory methods. In *Developing grounded theory*, pages 127–193. Routledge. Cited on pag. 82

Charmaz *et al.*(2012) Kathy Charmaz, Liska Belgrave *et al.* Qualitative interviewing and grounded theory analysis. *The SAGE handbook of interview research: The complexity of the craft*, 2:347–365. Cited on pag. 82

Choi(2013) Kyungsub S. Choi. Evaluating gender significance within a pair programming context. In *Proceedings of the 2013 46th Hawaii International Conference on System Sciences*, HICSS '13, pages 4817–4825, Washington, DC, USA. IEEE Computer Society. ISBN 978-0-7695-4892-0. doi: 10.1109/HICSS.2013.209. URL http://dx.doi.org/10.1109/HICSS.2013.209. Cited on pag. 27

Chong and Hurlbutt(2007) Jan Chong and Tom Hurlbutt. The social dynamics of pair programming. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 354–363, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2828-7. doi: 10.1109/ICSE.2007.87. URL http://dx.doi.org/10.1109/ICSE.2007.87. Cited on pag. 31

Christ *et al.*(2020) J. Christ, M. Huber and M. Pearl. *Remote Mob Programming: At Home, But Not Alone*. Amazon Digital Services LLC - KDP Print US. ISBN 9783982112602. URL https://books.google.com.br/books?id=bm92zQEACAAJ. Cited on pag. 45

Christiansen(2011) Ólavur Christiansen. The literature review in classic grounded theory studies: A methodological note. *The Grounded Theory Review*, 10(3):21–25. Cited on pag. 87

Clark and Watson(1995) Lee Anna Clark and David Watson. Constructing validity : Basic issues in objective scale development. Psychological Assessment 7(3):309-319. doi: 10.1037/1040-3590.7.3.309. Cited on pag. 80

Clarke(2003) Adele E Clarke. Situational analyses: Grounded theory mapping after the postmodern turn. *Symbolic interaction*, 26(4):553–576. Cited on pag. 72

Cline(2019) Alexander Cline. *Attribute clash: an archaeology of the ZX Spectrum*. Tese de Doutorado, Anglia Ruskin University. Cited on pag. 66

Cockburn and Williams(2000) Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. In *In eXtreme Programming and Flexible Processes in Software Engineering XP2000*, pages 223–247. Addison-Wesley. Cited on pag. 1, 31

Cockburn and Williams(2001) Alistair Cockburn and Laurie Williams. Extreme programming examined. chapter The Costs and Benefits of Pair Programming, pages 223–243. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-71040-4. URL http://dl.acm.org/citation.cfm?id=377517.377531. Cited on pag. 1, 30, 31

Constantine(1995) Larry L. Constantine. *Constantine on Peopleware*. Yourdon Press Computing Series. Prentice Hall Ptr, Englewood Cliffs, N.J., 1 edition edition. Paperback: 219 pages. Cited on pag. 1, 24, 30

Coplien and Harrison(2004) James O. Coplien and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. ISBN 0131467409. Cited on pag. 24, 36

Coplien and Schmidt(1995) James O. Coplien and Douglas C. Schmidt. *Pattern languages of program design*, pages 183–237. Addison-Wesley Professional, New York, NY, 1 edition edition. Cited on pag. 1, 24

Creswell(2008) John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publication, 3rd edition edition. Cited on pag. 71

Cronbach(1951) Lee J. Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334. ISSN 1860-0980. doi: 10.1007/BF02310555. URL https://doi.org/10.1007/BF02310555. Cited on pag. 80

da Silva and Agusti-Cullell(2008) Flavio Soares Correa da Silva and Agusti-Cullell. *Information Flow and Knowledge Sharing*. ISSN. Elsevier Science. ISBN 9780080569901. URL https://www.elsevier.com/books/information-flow-and-knowledge-sharing/james/978-0-444-52935-0. Cited on pag. xviii, xx, 17, 23, 54, 55, 56, 57, 96, 107, 114, 115, 116, 145, 167, 174, 177

da Silva Estacio and Prikladnicki(2015) Bernardo Jose da Silva Estacio and Rafael Prikladnicki. Distributed pair programming: A systematic literature review. *Information and Software Technology*, 63:1 – 10. ISSN 0950-5849. doi: http://dx.doi.org/10.1016/j.infsof.2015.02.011. URL http://www.sciencedirect.com/science/article/pii/S0950584915000476. Cited on pag. 28

Dagenais *et al.***(2010)** Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard and Jacqueline P. de Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 275–284, New York, NY, USA. Association for Computing Machinery. ISBN 9781605587196. doi: 10.1145/1806799.1806842. URL https://doi.org/10.1145/1806799.1806842. Cited on pag. 107

Dalton(2019) Jeff Dalton. *Mob Programming*, pages 193–194. Apress, Berkeley, CA. ISBN 978-1-4842-4206-3. doi: 10.1007/978-1-4842-4206-3_39. URL https://doi.org/10.1007/978-1-4842-4206-3_39. Cited on pag. 37

D'Angelo and Begel(2017) Sarah D'Angelo and Andrew Begel. Improving communication between pair programmers using shared gaze awareness. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6245–6290, New York, NY, USA. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025573. URL http://doi.acm.org/10.1145/3025453.3025573. Cited on pag. 29

de Aguiar(2003) Ademar Manuel Teixeira de Aguiar. *Framework documentation: A minimalist approach*. Doctoral thesis, Universidade do Porto (Portugal). Cited on pag. 138

de Oliveira Rosa and Goldman(2020) Thatiane de Oliveira Rosa and Alfredo Goldman. Is it possible to apply agile methods to contribute to the linux kernel? In Maria Paasivaara and Philippe Kruchten, editors, *Agile Processes in Software Engineering and Extreme Programming – Workshops*, pages 291–297, Cham. Springer International Publishing. ISBN 978-3-030-58858-8. Cited on pag. 6

de Sá(2022) Fernando Rodrigues de Sá. *H-Scrum : uma adaptacao de scrum para ambientes fortemente hierarquizados*. Doctoral thesis, Instituto Tecnologico de Aeronautica (ITA). Engenharia Eletronica e Computacao. Cited on pag. 11

de Sá(2023) Fernando Rodrigues de Sá. Scrum in strongly hierarchical organizations: A literature review. In Carla Rocha, Celio Santana Júnior, Fernando De Sá and Tiago Silva da Silva, editors, *Agile Methods*, pages 79–91, Cham. Springer International Publishing. ISBN 978-3-031-25648-6. Cited on pag. 11

de Sá *et al.*(2019a) Fernando Rodrigues de Sá, Everton Luiz de Resende Lucas and Adelmo Dias de Oliveira. Scrum in a strongly hierarchical organization. In Graziela Simone Tonin, Bernardo Estácio, Alfredo Goldman and Eduardo Guerra, editors, *Agile Methods*, pages 97–102, Cham. Springer International Publishing. ISBN 978-3-030-14310-7. Cited on pag. 10

de Sá *et al.*(2019b) Fernando Rodrigues de Sá, Ricardo Godoi Vieira and Adilson Marques da Cunha. Lessons learned from the agile transformation of an aeronautics computing center. In Paulo Meirelles, Maria Augusta Nelson and Carla Rocha, editors, *Agile Methods*, pages 85–91, Cham. Springer International Publishing. ISBN 978-3-030-36701-5. Cited on pag. 10

de Sá *et al.*(2022) Fernando Rodrigues de Sá, Ricardo Godoi Vieira and Adilson Marques da Cunha. Learning lessons from the scrum adoption in the brazilian air force. *Published by the IEEE Computer Society. IT Professional*, 24(1):49–55. doi: 10.1109/MITP.2021.3132310. Cited on pag. 11

DECEA(2011) DECEA. Roca 21-9. regulamento de centro de computação da aeronáutica, 2011. URL https://publicacoes.decea.mil.br/publicacao/roca-21-9. Cited on pag. 10

Deering and Williams(2020) Kris Deering and Jo Williams. Approaches to reviewing the literature in grounded theory: a framework. *Nurse Researcher*, 28(4). Cited on pag. 87

Deming(2018) W.E. Deming. *Out of the Crisis*. Mit Press. MIT Press. ISBN 9780262535946. URL https://books.google.com.br/books?id=7hb-tAEACAAJ. Cited on pag. 52

Denzin(2008) Norman K Denzin. *The landscape of qualitative research*, volume 1. Sage. Cited on pag. 72

Denzin and Lincoln(2011) Norman K Denzin and Yvonna S Lincoln. *The Sage handbook of qualitative research*. sage. Cited on pag. 72

DeVellis(2011) Robert F DeVellis. *Scale development : theory and applications*. Thousand Oaks SAGE Publications, Inc, 3rd ed edition. ISBN 9781452238418. Cited on pag. 78

di Bella *et al.*(2013) E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi and J. Vlasenko. Pair programming and software defects–a large, industrial case study. *IEEE Transactions on Software Engineering*, 39(7):930–953. ISSN 0098-5589. doi: 10.1109/TSE.2012.68. Cited on pag. 1, 30

Do Brasil(1988) Senado Federal Do Brasil. Constituição da república federativa do brasil. *Brasília: Senado Federal, Centro Gráfico*. Cited on pag. 9

Dragos(2021) Lucian Dragos. Mob vs pair : Comparing the two programming practices - a case study, 2021. Cited on pag. 38

Du *et al.*(2015) W. Du, M. Ozeki, H. Nomiya, K. Murata and M. Araki. Pair programming for enhancing communication in the fundamental c language exercise. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 664–665. doi: 10.1109/COMPSAC.2015.67. Cited on pag. 31

Dunne(2011) CiarÃ¡n Dunne. The place of the literature review in grounded theory research. *International Journal of Social Research Methodology*, 14(2):111–124. doi: 10.1080/13645579.2010.494930. URL https://doi.org/10.1080/13645579.2010.494930. Cited on pag. 87

Dvorak(2009) DL Dvorak. Final report: Nasa study on flight software complexity, 2009. Cited on pag. 53

Dyba *et al.*(2007) T. Dyba, E. Arisholm, D. I. K. Sjoberg, J. E. Hannay and F. Shull. Are two heads better than one? on the effectiveness of pair programming. *IEEE Software*, 24(6):12–15. ISSN 0740-7459. doi: 10.1109/MS.2007.158. Cited on pag. xxv, 2, 26, 27

Dyba and Dingsoyr(2008) Tore Dyba and Torgeir Dingsoyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2008.01.006. URL http://www.sciencedirect.com/science/article/pii/S0950584908000256. Cited on pag. 90

Easterbrook *et al.*(2008) Steve Easterbrook, Janice Singer, Margaret-Anne Storey and Daniela Damian. *Selecting Empirical Methods for Software Engineering Research*, pages 285–311. Springer London, London. ISBN 978-1-84800-044-5. doi: 10.1007/978-1-84800-044-5_11. URL https://doi.org/10.1007/978-1-84800-044-5_11. Cited on pag. 59

Easwaramoorthy and Zarinpoush(2006) Muthuswamy Easwaramoorthy and Fataneh Zarinpoush. Interviewing for research. *Imagine Canada*, 425. Cited on pag. 82

Elliott and Lazenbatt(2005) Naomi Elliott and Anne Lazenbatt. How to recognise a 'quality' grounded theory research study. *The Australian journal of advanced nursing : a quarterly publication of the Royal Australian Nursing Federation*, 22 3:48–52. Cited on pag. 61

Engel(2015) Gabriel Engel. Founder of rocket.chat, the leading free and open source team communication platform. Official WebSite, 2015. URL https://rocket.chat/press. Cited on pag. 7

Fenton and Pfleeger(1998) Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., USA, 2nd edition. ISBN 0534954251. Cited on pag. 53

Ferrario and Winter(2023) M. Ferrario and E. Winter. Applying human values theory to software engineering practice: Lessons and implications. *IEEE Transactions on Software Engineering*, 49(03):973–990. ISSN 1939-3520. doi: 10.1109/TSE.2022.3170087. Cited on pag. 15

Fink and Fielder(2003) A. Fink and E.P. Fielder. *The Survey Handbook*. Survey Kit Second Edition 1 1. SAGE Publications. ISBN 9780761925804. URL https://books.google.com.br/books?id=s_7GVySNWVgC. Cited on pag. 78

Freudenberg *et al.*(2007) S. Freudenberg, P. Romero and B. du Boulay. "talking the talk": Is intermediate-level conversation the key to the pair programming success story? In *Agile 2007 (AGILE 2007)*, pages 84–91. doi: 10.1109/AGILE.2007.1. Cited on pag. 31

Freudenberg and Wynne(2018) Sal Freudenberg and Matt Wynne. The surprisingly inclusive benefits of mob programming. In *Experience report*, XP '18. Cited on pag. 17, 40

Gay *et al.*(2021) P. Gay, L. Falco and CosmoQuest Build Team. Mob Programming Astronomy. In *American Astronomical Society Meeting Abstracts*, volume 53 of *American Astronomical Society Meeting Abstracts*, page 127.04. Cited on pag. 46

Georgieva and Allan(2008) Svetla Georgieva and George Allan. Best practices in project management through a grounded theory lens. *Electronic Journal of Business Research Methods*, 6(1). Cited on pag. 69, 85

GitLab(2018) GitLab. Oficial repositories of the source code. GitLab of Mezuro Project, 2018. URL http://gitlab.com/mezuro. Cited on pag. 6

Glaser(2002) Barney G Glaser. Constructivist grounded theory? *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 3(3). ISSN 1438-5627. doi: 10.17169/fqs-3.3.825. URL http://www.qualitative-research.net/index.php/fqs/article/view/825. Cited on pag. 72, 75

Glaser(1978) Barney G Glaser. *Theoretical sensitivity: advances in the methodology of grounded theory*. Advances in the methodology of grounded theory. Sociology Press. ISBN 9781884156014. URL https://books.google.com.br/books?id=73-2AAAAIAAJ. Cited on pag. 3, 61, 69, 70, 71, 72, 73, 75, 76, 77, 78, 84, 85, 86, 87, 88, 109, 110, 129, 131, 142, 143, 151, 153, 161, 163, 164, 165

Glaser(1998) Barney G Glaser. *Doing grounded theory : issues and discussions*. Mill Valley, CA : Sociology Press. ISBN 1884156118 9781884156113. Cited on pag. 71, 86, 89, 90

Glaser(1992) Barney G Glaser. *Emergence Vs Forcing: Basics of Grounded Theory Analysis*. Emergence vs. forcing. Sociology Press. ISBN 9781884156007. URL https://books.google.com.br/books?id=1ZJiQgAACAAJ. Cited on pag. 70, 71, 72, 84, 85, 88, 89

Glaser(2014) Barney G Glaser. *Memoing: A vital grounded theory procedure*. Sociology Press. Cited on pag. 86

Glaser(2016) Barney G Glaser. *The cry for help: Preserving autonomy doing GT research*. Sociology Press. Cited on pag. 72

Glaser and Holton(2004) Barney G. Glaser and Judith Holton. Remodeling grounded theory. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 5(2). doi: 10.17169/fqs-5.2.607. URL https://www.qualitative-research.net/index.php/fqs/article/view/607. Cited on pag. 61, 73

Glaser(2005) B.G. Glaser. *The Grounded Theory Perspective III: Theoretical Coding*. Sociology Press. ISBN 9781884156199. URL https://books.google.com.br/books?id=j3QnGwAACAAJ. Cited on pag. 70, 84, 85, 87, 88

Glaser *et al.*(1967) B.G. Glaser, A.L. Strauss and Sociology Press. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Observations (Chicago, Ill.). Aldine Publishing Company. ISBN 9780202300283. URL https://books.google.com.br/books?id=ppLZAAAAMAAJ. Cited on pag. 59, 70, 71, 75, 85, 87

Goldman and Santos(2019) Alfredo Goldman and Viviane Almeida Santos. Continuous improvement of an xp laboratory course: An 18 year history. In *Experience report*, Agile '19. Cited on pag. 6

Goldman *et al.*(2019) Alfredo Goldman, Thatiane de Oliveira Rosa and Viviane A. Santos. Having fun doing research on agile methods. In Paulo Meirelles, Maria Augusta Nelson and Carla Rocha, editors, *Agile Methods*, pages 147–164, Cham. Springer International Publishing. ISBN 978-3-030-36701-5. Cited on pag. 6

Goulding(2002) C. Goulding. *Grounded Theory: A Practical Guide for Management, Business and Market Researchers*. SAGE Publications. ISBN 9780761966838. URL https://books.google.com.br/books?id=O6EsGZjHm0wC. Cited on pag. 70

Grisham and Perry(2005) Paul S Grisham and Dewayne E Perry. Customer relationships and extreme programming. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–6. ACM. Cited on pag. 85

Group(2022) Science Museum Group. Sinclair ZX Spectrum+ microcomputer. *Science Museum Group. Sinclair ZX Spectrum. 2008-5038 Science Museum Group Collection Online. Accessed December 4, 2022. https://collection.sciencemuseumgroup.org.uk/objects/co8099426/sinclair-zx-spectrum-microcomputer-personal-computer.*, 4. Cited on pag. 67

Gruber(2007) Thomas Gruber. Ontology of folksonomy: A mash-up of apples and oranges. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(1):1–11. Cited on pag. 171

Gruber(1995) Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum. Comput. Stud.*, 43:907–928. Cited on pag. 57

Gubrium *et al.*(2002) J.F. Gubrium, J.A. Holstein, P.S.C.J.F. Gubrium, P.J.A. Holstein and inc Sage Publications. *Handbook of Interview Research: Context and Method*. A Sage reference title. SAGE Publications. ISBN 9780761919513. URL https://books.google.com.br/books?id=uQMUMQJZU4gC. Cited on pag. 71

Hall and Callery(2001) Wendy A Hall and Peter Callery. Enhancing the rigor of grounded theory: Incorporating reflexivity and relationality. *Qualitative health research*, 11(2):257–272. Cited on pag. 68

Hallberg(2006) Lillemor RM Hallberg. The "core category" of grounded theory: Making constant comparisons. *International journal of qualitative studies on health and well-being*, 1(3):141–148. Cited on pag. 74

Hannay *et al.*(2010) Jo E. Hannay, Erik Arisholm, Harald Engvik and Dag I. K. Sjoberg. Effects of personality on pair programming. *IEEE Trans. Softw. Eng.*, 36(1):61–80. ISSN 0098-5589. doi: 10.1109/TSE.2009.41. URL http://dx.doi.org/10.1109/TSE.2009.41. Cited on pag. 30, 31

Hardman and Hewson(1983) John Hardman and Andrew Hewson. *Die interne Struktur des ZX Spectrum*, pages 14–27. Birkhäuser Basel, Basel. ISBN 978-3-0348-6775-7. doi: 10.1007/978-3-0348-6775-7_2. URL https://doi.org/10.1007/978-3-0348-6775-7_2. Cited on pag. 67

Harsley *et al.*(2017) Rachel Harsley, Davide Fossati, Barbara Di Eugenio and Nick Green. Interactions of individual and pair programmers with an intelligent tutoring system for computer science. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 285–290, New York, NY, USA. ACM. ISBN 978-1-4503-4698-6. doi: 10.1145/3017680.3017786. URL http://doi.acm.org/10.1145/3017680.3017786. Cited on pag. 28

Heath and Cowley(2004) Helen Heath and Sarah Cowley. Developing a grounded theory approach: a comparison of glaser and strauss. *International journal of nursing studies*, 41(2):141–150. Cited on pag. 87

Hilton and Sankaranarayanan(2019) Michael Hilton and Sreecharan Sankaranarayanan. Online mob programming: Effective collaborative project-based learning. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 1283, New York, NY, USA. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3293774. URL https://doi.org/10.1145/3287324.3293774. Cited on pag. 44

Hoda(2021) Rashina Hoda. Decoding grounded theory for software engineering. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 326–327. doi: 10.1109/ICSE-Companion52605.2021.00139. Cited on pag. 60

Hoda(2022) Rashina Hoda. Socio-technical grounded theory for software engineering. *IEEE Transactions on Software Engineering*, 48(10):3808–3832. ISSN 1939-3520. doi: 10.1109/TSE.2021.3106280. Cited on pag. 60, 61

Hoda(2011) Rashina Hoda. Self-organizing agile teams: A grounded theory. Cited on pag. 86, 87

Hoda *et al.*(2010) Rashina Hoda, James Noble and Stuart Marshall. Using grounded theory to study the human aspects of software engineering. In *Human Aspects of Software Engineering*, HAoSE 10, New York, NY, USA. Association for Computing Machinery. ISBN 9781450305433. doi: 10.1145/1938595.1938605. URL https://doi.org/10.1145/1938595.1938605. Cited on pag. xxvi, 60, 69, 70

Hoda *et al.*(2011) Rashina Hoda, James Noble and Stuart Marshall. The impact of inadequate customer collaboration on self-organizing agile teams. *Information and Software Technology*, 53 (5):521 – 534. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2010.10.009. URL http://www.sciencedirect.com/science/article/pii/S0950584910001941. Special Section on Best Papers from XP2010. Cited on pag. xix, 85, 88, 89

Hoda *et al.*(2012) Rashina Hoda, James Noble and Stuart Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Softw. Engg.*, 17(6): 609–639. ISSN 1382-3256. doi: 10.1007/s10664-011-9161-0. URL https://doi.org/10.1007/s10664-011-9161-0. Cited on pag. xix, xxvi, 60, 69, 70, 76, 77

Hoda *et al.*(2013) Rashina Hoda, James Noble and Stuart Marshall. Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, 39(3):422–444. doi: 10.1109/TSE.2012.30. Cited on pag. 60

Höfer(2008) Andreas Höfer. Video analysis of pair programming. In *Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral*, APOS '08, pages 37–41, New York, NY, USA. ACM. ISBN 978-1-60558-021-0. doi: 10.1145/1370143.1370151. URL http://doi.acm.org/10.1145/1370143.1370151. Cited on pag. 31

Hohman and Slocum(2002) Moses M. Hohman and Andrew C. Slocum. Mob Programming and the Transition to XP. Relatório técnico. Cited on pag. 37, 38, 39, 51

Holton(2010) J Holton. The coding process and its challenges. the grounded theory review, 9 (1), 21–40, 2010. Cited on pag. xix, 72, 73

Hulkko and Abrahamsson(2005) Hanna Hulkko and Pekka Abrahamsson. A multiple case study on the impact of pair programming on product quality. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 495–504, New York, NY, USA. ACM. ISBN 1-58113-963-2. doi: 10.1145/1062455.1062545. URL http://doi.acm.org/10.1145/1062455.1062545. Cited on pag. 2, 26

Imai(2022) Saki Imai. Is github copilot a substitute for human pair-programming? an empirical study. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 319–321. doi: 10.1145/3510454.3522684. Cited on pag. 34

ISaaman and Tyler(1982) Daniel ISaaman and Jenny Tyler. *Programas De Jogos ESPACIAIS para ZX Spectrum, TKs, CP200, Apple, & TRS80 (Original English Title: COMPUTER SPACEGAMES)*. Lutecia LTDA. ISBN 85-19-23219-1. Cited on pag. xviii, xxiv, 65, 253

Isaaman *et al.*(1982) Daniel Isaaman, Rex Archer and Jenny Tyler. *Computer battlegames:[for ZX Spectrum, ZX 81, BBC, TRS-80, Apple, VIC & PET]*. Usborne Publishing Ltd. Cited on pag. xviii, xxiv, 65, 253

Issaee *et al.*(2021) Arash Issaee, Renate Motschnig and Oswald Comber. Pair- versus solo-programming of mini-games as a setting for learning to program: An action research approach. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. doi: 10.1109/FIE49875.2021.9637178. Cited on pag. 28

Issaee *et al.*(2022) Arash Issaee, Renate Motschnig and Katrin Goltl. Learning to program in secondary classrooms: Students' and teachers' perceptions of the pair-programming setting. In *2022 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. doi: 10.1109/FIE56618.2022.9962635. Cited on pag. 28

Johnson and Johnson(1984) David W. Johnson and Roger T. Johnson. Structuring groups for cooperative learning. *Organizational Behavior Teaching Review*, 9(4):8–17. doi: 10.1177/105256298400900404. URL https://doi.org/10.1177/105256298400900404. Cited on pag. 44

Jones and Fleming(2013) D. L. Jones and S. D. Fleming. What use is a backseat driver? a qualitative investigation of pair programming. In *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, pages 103–110. doi: 10.1109/VLHCC.2013.6645252. Cited on pag. 32

Kadam and Nayak(2016) Vinod V. Kadam and Rajkishore Nayak. *Basics of Acoustic Science*, pages 33–42. Springer Singapore, Singapore. ISBN 978-981-10-1476-5. doi: 10.1007/978-981-10-1476-5_2. URL https://doi.org/10.1007/978-981-10-1476-5_2. Cited on pag. 120

Kaieda *et al.*(2020) Shota Kaieda, Daisuke Saito, Hironori Washizaki and Yoshiaki Fukasawa. Fusion of mob programming and student mentor methods in programming classes. In *82nd Proceedings of the National Conference*, volume 2020, pages 705–706. Cited on pag. 45

Kaieda *et al.*(2021) Shota Kaieda, Daisuke Saito, Hironori Washizaki and Yoshiaki Fukazawa. Work-in-progress: Analysis of the use of mentoring with online mob programming. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 1424–1428. doi: 10.1109/EDUCON46332.2021.9454001. Cited on pag. 44

KalibroConfigurations(2018) KalibroConfigurations. Kalibro configurations, a source code analysis configuration web service. GitLab of Kalibro Configurations, which is also part of the Mezuro project., 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/configurations/README.md. Cited on pag. 6

KalibroProcessor(2018) KalibroProcessor. Oficial kalibro processor source code repository. GitLab of Kalibro Processor part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/processor/README.rdoc. Cited on pag. 6

Kan and Parry(2004) Melanie M Kan and Ken W Parry. Identifying paradox: A grounded theory of leadership in overcoming resistance to change. *The Leadership Quarterly*, 15(4):467–491. Cited on pag. 88, 109, 110, 129, 131, 142, 143, 151, 153, 161, 163, 164, 165

Kattan(2016a) Herez Moise Kattan. Wiki of CCSL, 2016a. URL http://ccsl.ime.usp.br/wiki/SwarmQuestionnaire. Cited on pag. 80

Kattan(2019a) Herez Moise Kattan. Software development practices patterns: from pair to mob programming. In *Proceedings of the Regional School of Software Engineering (ERES). Anais da III Escola Regional de Engenharia de Software*, pages 147–156, Porto Alegre, RS, Brasil. Brazilian Computing Society - SBC. URL https://sol.sbc.org.br/index.php/eres/article/view/8507. Cited on pag. 1, 14

Kattan(2015) Herez Moise Kattan. Programming and review simultaneous in pairs: a pair programming extension. Master Thesis, Institute for Technological Research of the Sao Paulo State, Brazil. URL cassiopea.ipt.br/teses/2015_EC_Herez_Moise_Kattan.pdf. Cited on pag. 1, 23, 27, 36, 52

Kattan(2016b) Herez Moise Kattan. Illuminated arrow: a research method to software engineering based on action research, systematic review and grounded theory. In *13th International Conference on Information Systems and Technology Management*. Paper submission in 1 Dec 2015. Presented at Session4A. Pages: 1971-1978. Sao Paulo, SP, Brazil. Cited on pag. 13, 62, 63

Kattan(2017a) Herez Moise Kattan. Those who fail to learn from history are doomed to repeat it. In *Agile Processes in Software Engineering and Extreme Programming: poster presented at the 18th International Conference on Agile Software Development, XP 2017, held in Cologne, Germany, in May 22-26*. doi: 10.13140/RG.2.2.20864.02563. Cited on pag. 13, 62, 63

Kattan(2017b) Herez Moise Kattan. Pares de policiais para seguranca, pares de programadores para qualidade. In *Video recorded at The Developers Conference 2017*, Sao Paulo, SP, Brazil. TDC 2017. The Developers Conference - Agile. July 22, 2017. URL https://thedevconf.com/tdconline. Cited on pag. 13

Kattan(2018a) Herez Moise Kattan. Theory of altruism on software development practices patterns. In *Proceedings of the 19th International Conference on Agile Software Development: Companion*, XP '18, New York, NY, USA. Association for Computing Machinery. ISBN 9781450364225. doi: 10.1145/3234152.3314991. URL https://doi.org/10.1145/3234152.3314991. Cited on pag. 14

Kattan(2018b) Herez Moise Kattan. Swarm or pair? strengths and weaknesses of pair programming and mob programming). In *Agile Processes in Software Engineering and Extreme Programming: Poster presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25*. Accessed: July 2018. Cited on pag. 1, 14

Kattan(2018c) Herez Moise Kattan. To swarm or not to swarm (programming and review simultaneous in pairs). In *Agile Processes in Software Engineering and Extreme Programming: WorkShop presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25*. Accessed: July 2018. Cited on pag. 2, 14

Kattan(2019b) Herez Moise Kattan. Mob programming and simultaneous style pair programming in the development of a battle royale game: An action research. In Paulo Meirelles, Maria Augusta Nelson and Carla Rocha, editors, *Agile Methods*, pages 41–57, Cham. Springer International Publishing. ISBN 978-3-030-36701-5. Cited on pag. 2, 6, 14

Kattan and Goldman(2017) Herez Moise Kattan and Alfredo Goldman. Software development practices patterns. In Hubert Baumeister, Horst Lichter and Matthias Riebisch, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 298–303, Cham. Springer International Publishing. ISBN 978-3-319-57633-6. Cited on pag. 13, 39

Kattan *et al.*(2018a) Herez Moise Kattan, Frederico Oliveira, Alfredo Goldman and Joseph William Yoder. Mob programming: The state of the art and three case studies of open source software. In Viviane Almeida dos Santos, Gustavo Henrique Lima Pinto and Adolfo Gustavo Serra Seca Neto, editors, *Agile Methods*, pages 146–160, Cham. Springer International Publishing. ISBN 978-3-319-73673-0. Cited on pag. xvii, 2, 6, 14, 39, 40, 41, 51, 61, 62

Kattan *et al.*(2018b) Herez Moise Kattan, Flavio Soares, Alfredo Goldman, Eduardo Deboni and Eduardo Guerra. Swarm or pair? strengths and weaknesses of pair programming and mob programming). In *Agile Processes in Software Engineering and Extreme Programming: Short Paper presented at the 19th International Conference on Agile Software Development, XP 2018, held in Porto, Portugal, on May 21-25*. Proceedings of XP'18 Companion, Porto, Portugal. doi: 10.1145/3234152.3234169. May 21-25. Cited on pag. 1, 14, 62

Kayes(2015) Gillyanne Kayes. *Singing and the Actor*. Routledge. Cited on pag. 120

Keeling and Runde(2019) Michael Keeling and Joe Runde. Harvesting mob programming patterns: Observing how we work. *Experience Report. Aug 2019, Agile2019 Conference. Copyright 2019 is held by the authors.* Cited on pag. xviii, xxv, 49, 50, 51

Kenealy(2008) Graham J.J. Kenealy. Management research and grounded theory: A review of grounded theorybuilding approach in organisational and management research. *Grounded Theory Review An international journal*, 07. Cited on pag. 60

Kenny and Fourie(2015) Méabh Kenny and Robert Fourie. Contrasting classic, straussian, and constructivist grounded theory: Methodological and philosophical conflicts. *The Qualitative Report*. Cited on pag. xix, 71, 72, 73, 74, 75, 76

Kitchenham and Charters(2007) Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Relatório Técnico EBSE 2007-001, Keele University and Durham University Joint Report. URL https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf. Cited on pag. 59, 61

Kitchenham *et al.*(2010) Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi and Stephen Linkman. Systematic literature reviews in software engineering - a tertiary study. *Inf. Softw. Technol.*, 52(8):792–805. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.03.006. URL http://dx.doi.org/10.1016/j.infsof.2010.03.006. Cited on pag. 60

Kitchenham(2012) Barbara A. Kitchenham. Systematic review in software engineering: Where we are and where we should be going. In *Proceedings of the 2Nd International Workshop on Evidential Assessment of Software Technologies*, EAST '12, pages 1–2, New York, NY, USA. ACM. ISBN 978-1-4503-1509-8. doi: 10.1145/2372233.2372235. URL http://doi.acm.org/10.1145/2372233.2372235. Cited on pag. 60, 61

Kitchenham *et al.*(2004) Barbara A. Kitchenham, Tore Dyba and Magne Jorgensen. Evidence-based software engineering. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 273–281, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2163-0. URL http://dl.acm.org/citation.cfm?id=998675.999432. Cited on pag. xxv, xxvi, 62, 63, 90

Kock(2010) N. Kock. *Information Systems Action Research: An Applied View of Emerging Concepts and Methods*. Integrated Series in Information Systems. Springer US. ISBN 9781441942104. URL https://books.google.com.br/books?id=N-f9ngEACAAJ. Cited on pag. 59

Kock(2004) Ned Kock. The three threats of action research: A discussion of methodological antidotes in the context of an information systems study. *Decis. Support Syst.*, 37(2):265–286. ISSN 0167-9236. doi: 10.1016/S0167-9236(03)00022-8. URL http://dx.doi.org/10.1016/S0167-9236(03)00022-8. Cited on pag. 59

Kolekti(2018) Kolekti. Oficial kolekti source code repository. GitLab of Kolekti part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/kolekti/README.md. Cited on pag. 6

KolektiAnalizo(2018) KolektiAnalizo. Generic parser for analizo static code metrics collector. GitLab of KolektiAnalizo part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/kolekti_analizo/README.md. Cited on pag. 6

KolektiCCPHPMD(2018) KolektiCCPHPMD. Generic parser for kolekti kolekticcphpmd static code metrics collector. GitLab of Kolekti KolektiCCPHPMD, part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/kolekti_cc_phpmd/README.md. Cited on pag. 6

KolektiMetricFu(2018) KolektiMetricFu. Generic parser for kolekti metricfu static code metrics collector. GitLab of Kolekti Metric Fu, part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/kolekti_metricfu/README.md. Cited on pag. 6

KolektiRadon(2018) KolektiRadon. Generic parser for kolekti radon static code metrics collector. GitLab of Kolekti Radon, part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/kalibro/kolekti_radon/README.md. Cited on pag. 6

Laughlin *et al.***(2006)** Patrick R Laughlin, Erin C Hatch, Jonathan S Silver and Lee Boh. Groups perform better than the best individuals on letters-to-numbers problems: effects of group size. *Journal of Personality and social Psychology*, 90(4):644. Cited on pag. 44, 129, 145, 146

Lean(2016) Tom Lean. *Electronic dreams: How 1980s Britain Learned to love the computer*. Bloomsbury Publishing. Cited on pag. 65

Lempert(2007) Lora Bex Lempert. Asking questions of the data: Memo writing in the grounded theory tradition. In *The SAGE Handbook of Grounded Theory*, pages 245–264. SAGE Publications Ltd, 1 Oliver's Yard, 55 City Road, London England EC1Y 1SP United Kingdom. Cited on pag. 90

Li *et al.***(2013)** Zhen Li, C. Plaue and E. Kraemer. A spirit of camaraderie: The impact of pair programming on retention. In *2013 26th International Conference on Software Engineering Education and Training (CSEE T)*, pages 209–218. doi: 10.1109/CSEET.2013.6595252. Cited on pag. 29

Lilienthal(2017) Carola Lilienthal. *From Pair Programming to Mob Programming to Mob Architecting*, pages 3–12. Springer International Publishing, Cham. ISBN 978-3-319-49421-0. doi: 10.1007/978-3-319-49421-0_1. URL https://doi.org/10.1007/978-3-319-49421-0_1. Cited on pag. 39

Lindskog and Netz(2021) Carin Lindskog and Johan Netz. Balancing between stability and change in agile teams. *International Journal of Managing Projects in Business*, 14(7):1529–1554. ISSN 1753-8378. doi: 10.1108/IJMPB-12-2020-0366. URL https://doi.org/10.1108/IJMPB-12-2020-0366. Cited on pag. 60

Lisa M. Given(2008) John W. Creswell et al. Lisa M. Given, Kristie Saumure. *The Sage Encyclopedia of Qualitative Research Methods. General editor: Lisa M. Given*, volume 1–2 of *A Sage Reference Publication*. SAGE Publications, Inc; 1 edition (August 21, 2008). ISBN 9781412941631. doi: 10.4135/9781412963909. Cited on pag. 71

Litwin(1995) M.S. Litwin. *How to Measure Survey Reliability and Validity*. SAGE Publications. ISBN 9781506319810. URL https://books.google.com.br/books?id=qN0sCgAAQBAJ. Cited on pag. 80, 81

Lucian(2017) Chris Lucian. Growing the mob. *Experience Report. Aug 2017, Agile2017 Conference. Copyright 2017 is held by Chris Lucian*. Cited on pag. xviii, 46, 47, 48

Lucian(2021) Chris Lucian. Improving technical quality with mob programming and collective habits. *Talk with video available at Agile Alliance WebSite. July 21, 2021, Agile2021 Conference. Copyright 2021 is held by Chris Lucian*. Cited on pag. 38

Madeyski and Kitchenham(2017) Lech Madeyski and Barbara Kitchenham. Would wider adoption of reproducible research be beneficial for empirical software engineering research? *Journal of Intelligent & Fuzzy Systems*, 32:1509–1521. ISSN 1875-8967. doi: 10.3233/JIFS-169146. URL https://doi.org/10.3233/JIFS-169146. 2. Cited on pag. 61

Mandhan *et al.***(2015)** N. Mandhan, D. K. Verma and S. Kumar. Analysis of approach for predicting software defect density using static metrics. In *International Conference on Computing, Communication Automation*, pages 880–886. doi: 10.1109/CCAA.2015.7148499. Cited on pag. 52

Marecki and Yerzmyey(2019) Piotr Marecki and Yerzmyey. ZX spectrum scene poetry: genre, platform, localness. *Digital Scholarship in the Humanities*, 35(3):623–634. ISSN 2055-7671. doi: 10.1093/llc/fqz057. URL https://doi.org/10.1093/llc/fqz057. Cited on pag. 66

Martin *et al.***(2009)** Angela Martin, Robert Biddle and James Noble. The xp customer team: A grounded theory. In *2009 Agile Conference*, pages 57–64. doi: 10.1109/AGILE.2009.70. Cited on pag. 60

Martin *et al.***(2010)** Angela Martin, Robert Biddle and James Noble. *An Ideal Customer: A Grounded Theory of Requirements Elicitation, Communication and Acceptance on Agile Projects*, pages 111–141. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-12575-1. doi: 10.1007/978-3-642-12575-1_6. URL https://doi.org/10.1007/978-3-642-12575-1_6. Cited on pag. 60

Masood *et al.***(2022)** Zainab Masood, Rashina Hoda and Kelly Blincoe. Real world scrum a grounded theory of variations in practice. *IEEE Transactions on Software Engineering*, 48(5): 1579–1591. doi: 10.1109/TSE.2020.3025317. Cited on pag. 61

Mayo(1933) Elton Mayo. The hawthorne experiment. western electric company. *2016). Classics of organization theory*, pages 134–141. Cited on pag. 108, 173

McAlpine(2017) Kenneth B. McAlpine. The sound of 1-bit: technical constraint and musical creativity on the 48k sinclair zx spectrum. *GAME: The Italian Journal of Game Studies*, 1(6). ISSN 2280-7705. Cited on pag. 65

Mccallin(2003) Antoinette Mccallin. Grappling with the literature in a grounded theory study. *Contemporary Nurse*, 15(1-2):61–69. doi: 10.5172/conu.15.1-2.61. URL https://doi.org/10.5172/conu.15.1-2.61. PMID: 14649509. Cited on pag. 87

McChesney(2016) Ian McChesney. Three years of student pair programming: Action research insights and outcomes. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 84–89, New York, NY, USA. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844565. URL http://doi.acm.org/10.1145/2839509.2844565. Cited on pag. 27

McDowell *et al.***(2006)** Charlie McDowell, Linda Werner, Heather E. Bullock and Julian Fernald. Pair programming improves student retention, confidence, and program quality. *Commun. ACM*, 49(8):90–95. ISSN 0001-0782. doi: 10.1145/1145287.1145293. URL http://doi.acm.org/10.1145/1145287.1145293. Cited on pag. 2, 27

Melo(2013) Claudia de Oliveira Melo. *Productivity of agile teams: an empirical evaluation of factors and monitoring processes.* Doctoral thesis, Institute of Mathematics and Statistcs, University of Sao Paulo, Brazil. doi: 10.11606/T.45.2013.tde-25052015-120242. URL https://www.teses.usp.br/teses/disponiveis/45/45134/tde-25052015-120242/. Cited on pag. 53

Menzies *et al.***(2003)** Tim Menzies, Jefferey Smith and David Raffo. When is pair programming better? Website of the author, 2003. URL http://menzies.us/pdf/04pairprog.pdf. Cited on pag. xvii, 25

MetricFu(2018) MetricFu. Metricfu static code metrics collector. GitHub of Metric Fu, 2018. URL https://github.com/metricfu/metric_fu. Cited on pag. 6

Mezuro.org(2018) Mezuro.org. Stable implementation of the mezuro project, it is possible to create repositories or monitor the metrics of the existing ones. WebSite of Mezuro Project, 2018. URL http://Mezuro.org. Cited on pag. 6

Miller(1985) Judith Miller. *Beginning BASIC with the ZX Spectrum*, pages 56–64. Macmillan Education UK, London. ISBN 978-1-349-81211-0. doi: 10.1007/978-1-349-81211-0. URL https://doi.org/10.1007/978-1-349-81211-0. Cited on pag. 65

Missiroli *et al.***(2016)** Marcello Missiroli, Daniel Russo and Paolo Ciancarini. Learning agile software development in high school: An investigation. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 293–302, New York, NY, USA. ACM. ISBN 978-1-4503-4205-6. doi: 10.1145/2889160.2889180. URL http://doi.acm.org/10.1145/2889160.2889180. Cited on pag. 28

Morse *et al.***(2016)** Janice M Morse, Barbara J Bowers, Kathy Charmaz, Juliet Corbin, Adele E Clarke and Phyllis Noerager Stern. *Developing grounded theory: The second generation*, volume 3. Routledge. Cited on pag. 72

Morse(1994) J.M. Morse. *Critical Issues in Qualitative Research Methods*. SAGE Publications. ISBN 9780803950436. URL https://books.google.co.uk/books?id=3ZlrPlpU1oAC. Cited on pag. 71

Muller and Padberg(2004) Matthias M. Muller and Frank Padberg. An empirical study about the feelgood factor in pair programming. In *Proceedings of the Software Metrics, 10th International Symposium*, METRICS '04, pages 151–158, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-2129-0. doi: 10.1109/METRICS.2004.8. URL http://dx.doi.org/10.1109/METRICS.2004.8. Cited on pag. 27

Müter(2018) LHF Müter. User stories in an integrated development environment. Master Thesis. Cited on pag. 38

Myers(2019) Michael D Myers. *Qualitative research in business and management*. Sage. Cited on pag. 63

Myers and Avison(2002) Michael D. Myers and David Avison. An introduction to qualitative research in information systems. In *Qualitative research in information systems: a reader*, pages 3–12. SAGE Publications. URL https://eprints.soton.ac.uk/35818/. Cited on pag. 63

Nathaniel(2019) A. Nathaniel. How classic grounded theorists teach the method. *Grounded Theory Review*, 18(1). URL http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-103787. Cited on pag. 60, 61

Nathaniel(2022) Alvita Nathaniel. When and how to use extant literature in classic grounded theory. *American Journal of Qualitative Research*, 6(2):45–59. Cited on pag. 60, 61

Nicolaescu *et al.***(2015)** Ana Nicolaescu, Horst Lichter and Yi Xu. Evolution of object oriented coupling metrics: A sampling of 25 years of research. In *Proceedings of the Second International Workshop on Software Architecture and Metrics*, SAM '15, pages 48–54, Piscataway, NJ, USA. IEEE Press. URL http://dl.acm.org/citation.cfm?id=2821327.2821337. Cited on pag. 52

Noori and Kazemifard(2015) F. Noori and M. Kazemifard. Simulation of pair programming using multi-agent and mbti personality model. In *2015 Sixth International Conference of Cognitive Science (ICCS)*, pages 29–36. doi: 10.1109/COGSCI.2015.7426665. Cited on pag. 30

Nosek(1998) John T. Nosek. The case for collaborative programming. *Commun. ACM*, 41(3): 105–108. ISSN 0001-0782. doi: 10.1145/272287.272333. URL http://doi.acm.org/10.1145/272287.272333. Cited on pag. 2, 24, 30

Oliver(2012) Carolyn Oliver. Critical realist grounded theory: A new approach for social work research. *The British Journal of Social Work*, 42(2):371–387. ISSN 00453102, 1468263X. URL http://www.jstor.org/stable/43771640. Cited on pag. 74

Onions(2006) Patrick E.W. Onions. Grounded theory applications in reviewing knowledge management literature patrick e.w. onions the knowledge studio, 2006. Cited on pag. 73

Opie(2004) C. Opie. *Doing Educational Research*. SAGE Publications. ISBN 9781446233566. URL https://books.google.com.br/books?id=vf7z00J4owcC. Cited on pag. 60

Orlikowski and Baroudi(1991) Wanda J. Orlikowski and Jack J. Baroudi. Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research*, 2(1):1–28. ISSN 10477047, 15265536. URL http://www.jstor.org/stable/23010611. Cited on pag. 63

Padberg and Muller(2003) F. Padberg and M. M. Muller. Analyzing the cost and benefit of pair programming. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No.03EX717)*, pages 166–177. doi: 10.1109/METRIC.2003.1232465. Cited on pag. 25, 31

Padberg and Müller(2003) Frank Padberg and Matthias M. Müller. Analyzing the cost and benefit of pair programming. In *Proceedings of the 9th International Symposium on Software Metrics*, METRICS '03, pages 166–, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-1987-3. URL http://dl.acm.org/citation.cfm?id=942804.943771. Cited on pag. 25, 31

Padhy *et al.*(2015) N. Padhy, R. Panigrahi and S. Baboo. A systematic literature review of an object oriented metric: Reusability. In *2015 International Conference on Computational Intelligence and Networks*, pages 190–191. doi: 10.1109/CINE.2015.44. Cited on pag. 52

Parrish *et al.*(2004) Allen Parrish, Randy Smith, David Hale and Joanne Hale. A field study of developer pairs: Productivity impacts and implications. *IEEE Softw.*, 21(5):76–79. ISSN 0740-7459. doi: 10.1109/MS.2004.1331306. URL http://dx.doi.org/10.1109/MS.2004.1331306. Cited on pag. 26

Parry(1998) Ken W Parry. Grounded theory and social process: A new direction for leadership research. *The leadership quarterly*, 9(1):85–105. Cited on pag. 90, 170

Pearl(2018) Mark Pearl. Code with the wisdom of the crowd: Get better together with mob programming. *Code with the Wisdom of the Crowd*, pages 1–100. Cited on pag. 37

PHPMD(2018) PHPMD. Phpmd static code metrics collector. WebSite of PHPMD, 2018. URL https://phpmd.org/. Cited on pag. 6

Picard(2012) Janice Marie Picard. *A Grounded Theory of "Not Pulling Weight" In Online MBA Group Work*. Lancaster University (United Kingdom). Cited on pag. 88

Pithon(2004) Antonio Jose Caulliraux Pithon. *Projeto organizacional para a engenharia concorrente no ambito das empresas virtuais*. Doctoral thesis, Escola de Engenharia da Universidade do Minho Departamento de ProduÃ§Ã£o e Sistemas, Portugal. URL http://hdl.handle.net/1822/897. Cited on pag. 36

Plonka and van der Linden(2012) Laura Plonka and Janet van der Linden. Why developers don't pair more often. In *Proceedings of the 5th International Workshop on Co-operative and*

Human Aspects of Software Engineering, CHASE '12, pages 123–125, Piscataway, NJ, USA. IEEE Press. ISBN 978-1-4673-1824-2. URL http://dl.acm.org/citation.cfm?id=2663638.2663664. Cited on pag. 32

Plonka *et al.***(2012)** Laura Plonka, Helen Sharp and Janet van der Linden. Disengagement in pair programming: Does it matter? In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 496–506, Piscataway, NJ, USA. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337282. Cited on pag. 31

Plonka *et al.***(2015)** Laura Plonka, Helen Sharp, Janet van der Linden and Yvonne Dittrich. Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human-Computer Studies*, 73:66 – 78. ISSN 1071-5819. doi: http://dx.doi.org/10.1016/j.ijhcs.2014.09.001. URL http://www.sciencedirect.com/science/article/pii/S1071581914001207. Cited on pag. 1, 32

Plowman(2017) Gary Plowman. *ZX Spectrum Games Code Club: Twenty fun games to code and learn; 1*. Gazzapper Press. ISBN 9780993474453. URL https://bib-pubdb1.desy.de/record/344643. Cited on pag. 65

Poonam and Yasser(2018) R. Poonam and C. M. Yasser. An experimental study to investigate personality traits on pair programming efficiency in extreme programming. In *2018 5th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 95–99. doi: 10.1109/IEA.2018.8387077. Cited on pag. 30

Powell and Anslow(2019) Torrey Powell and Craig Anslow. Joining the mob at clearlink. *Experience Report. May 2019, XP 2019 Conference. Copyright 2019 is held by the authors.* Cited on pag. 46

Prezento(2018) Prezento. Oficial prezento source code repository. GitLab of Prezento part of Mezuro Project, 2018. URL https://gitlab.com/mezuro/mezuro/blob/master/prezento/README.md. Cited on pag. 6

Pyhäjärvi and Falco(2016) Maaret Pyhäjärvi and Llewellyn Falco. The mob programming guidebook, 2016. Cited on pag. 38

Radon(2018) Radon. Radon static code metrics collector. WebSite of Radon, 2018. URL https://pypi.org/project/radon/. Cited on pag. 6

Rajendran Swamidurai and Kannan(2014) Brad Dennis Rajendran Swamidurai and Uma Kannan. Investigating the impact of peer code review and pair programming on test-driven development. In *IEEE SOUTHEASTCON 2014*, SOUTHEASTCON '14, Conference Location: Lexington, KY, USA. IEEE. ISBN 978-1-4799-6585-4. doi: 10.1109/SECON.2014.6950664. Cited on pag. 2, 23

Rajpal(2018) Mark Rajpal. Effective distributed pair programming. In *Proceedings of the 13th International Conference on Global Software Engineering*, ICGSE '18, pages 6–10, New York, NY, USA. Association for Computing Machinery. ISBN 9781450357173. doi: 10.1145/3196369.3196388. URL https://doi.org/10.1145/3196369.3196388. Cited on pag. 29

Rashi Agarwal(2020) Shashank Sahu Rashi Agarwal, Supriya Khaitan. Intelligent agents. *Distributed artificial intelligence: A modern approach*. Cited on pag. 38

Rodríguez *et al.***(2017)** Fernando J. Rodríguez, Kimberly Michelle Price and Kristy Elizabeth Boyer. Exploring the pair programming process: Characteristics of effective collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 507–512, New York, NY, USA. ACM. ISBN 978-1-4503-4698-6. doi: 10.1145/3017680.3017748. URL http://doi.acm.org/10.1145/3017680.3017748. Cited on pag. 28

Rogers *et al.*(**2021**) Jen Rogers, Austin H Patton, Luke Harmon, Alexander Lex and Miriah Meyer. Insights from experiments with rigor in an evobio design study. *IEEE Transactions on Visualization and Computer Graphics (InfoVis)*, 27(2):1106–1116. doi: 10.1109/TVCG.2020.3030405. Cited on pag. 63

Rooddehghan *et al.*(**2019**) Zahra Rooddehghan, Zohreh ParsaYekta and Alireza N Nasrabadi. Equity in nursing care: A grounded theory study. *Nursing Ethics*, 26(2):598–610. doi: 10.1177/0969733017712079. URL https://doi.org/10.1177/0969733017712079. PMID: 28670955. Cited on pag. 89

Roque Hernandez *et al.*(**2020**) Ramon Ventura Roque Hernandez, Sergio Armando Guerra Moya and Adan Lopez Mendoza. Solo, pair or Mob programming: Which should be used in university? *Apertura*, 12(1). ISSN 16656180, 20071094. doi: 10.32870/Ap.v12n1.1791. URL http://www.udgvirtual.udg.mx/apertura/index.php/apertura/article/view/1791. Cited on pag. 42

Ruvalcaba *et al.*(**2016**) Omar Ruvalcaba, Linda Werner and Jill Denner. Observations of pair programming: Variations in collaboration across demographic groups. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 90–95, New York, NY, USA. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844558. URL http://doi.acm.org/10.1145/2839509.2844558. Cited on pag. 27

Safety *et al.*(**2020**) Occupational Safety, Health Administration *et al.* Guidance on preparing workplaces for covid-19. *US: Department of Labor*. Cited on pag. 122, 123, 124

Saltz and Shamshurin(2017) J. S. Saltz and I. Shamshurin. Does pair programming work in a data science context? an initial case study. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2348–2354. doi: 10.1109/BigData.2017.8258189. Cited on pag. 33

Sawyer *et al.*(**2010**) Steve Sawyer, Patricia J. Guinan and Jay Cooprider. Social interactions of information systems development teams: a performance perspective. *Information Systems Journal*, 20(1):81–107. doi: https://doi.org/10.1111/j.1365-2575.2008.00311.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2575.2008.00311.x. Cited on pag. 107, 108

Schöpfel(2010) Joachim Schöpfel. Towards a Prague Definition of Grey Literature. In *Twelfth International Conference on Grey Literature: Transparency in Grey Literature. Grey Tech Approaches to High Tech Issues. Prague, 6-7 December 2010*, pages 11–26, Czech Republic. URL https://archivesic.ccsd.cnrs.fr/sic_00581570. Cited on pag. 62, 63

Seyam and McCrickard(2015) M. Seyam and S. McCrickard. Collaborating on mobile app design through pair programming: A practice-oriented approach overview and expert review. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pages 124–131. doi: 10.1109/CTS.2015.7210412. Cited on pag. 32

Seyam and McCrickard(2016) Mohammed Seyam and D. Scott McCrickard. Teaching mobile development with pair programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 96–101, New York, NY, USA. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844644. URL http://doi.acm.org/10.1145/2839509.2844644. Cited on pag. 32

Shiraishi *et al.*(**2019**) Makoto Shiraishi, Hironori Washizaki, Yoshiaki Fukazawa and Joseph Yoder. Mob programming: A systematic literature review. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 616–621. doi: 10.1109/COMPSAC.2019.10276. Cited on pag. 38

Shiraishi *et al.*(**2021**) Makoto Shiraishi, Hironori Washizaki, Daisuke Saito and Yoshiaki Fukazawa. Comparing participants' brainwaves during solo, pair, and mob programming. In

Peggy Gregory, Casper Lassenius, Xiaofeng Wang and Philippe Kruchten, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 200–209, Cham. Springer International Publishing. ISBN 978-3-030-78098-2. Cited on pag. xviii, 42, 43, 44

Sillitti *et al.***(2012)** Alberto Sillitti, Giancarlo Succi and Jelena Vlasenko. Understanding the impact of pair programming on developers attention: A case study on a large industrial experimentation. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1094–1101, Piscataway, NJ, USA. IEEE Press. ISBN 978-1-4673-1067-3. URL http://dl.acm.org/citation.cfm?id=2337223.2337366. Cited on pag. 32

Sjoberg and Bergersen(2023) D. K. Sjoberg and G. Bergersen. Construct validity in software engineering. *IEEE Transactions on Software Engineering*, 49(03):1374–1396. ISSN 1939-3520. doi: 10.1109/TSE.2022.3176725. Cited on pag. 91

Sjoberg *et al.***(2007)** Dag I. K. Sjoberg, Tore Dyba and Magne Jorgensen. The future of empirical methods in software engineering research. In *2007 Future of Software Engineering*, FOSE '07, page 358?378, USA. IEEE Computer Society. ISBN 0769528295. doi: 10.1109/FOSE.2007.30. URL https://doi.org/10.1109/FOSE.2007.30. Cited on pag. 59

Smith and Menzies(2002) Jefferey Smith and Tim Menzies. Should nasa embrace agile processes. Website of the author, 2002. Cited on pag. 31

Spectrum(2022) Museum LOAD ZX Spectrum. A museum dedicated to our first personal computer, 2022. Cited on pag. 67

Sreecharan Sankaranarayanan(2019) Xu Wang Sreecharan Sankaranarayanan. Online mob programming: Bridging the 21st century workplace and the classroom. *Proceedings of Computer-Supported Collaborative Learning*, 2. URL https://par.nsf.gov/biblio/10104106. Cited on pag. 44

Stachniak(2015) Zbigniew Stachniak. Red clones: The soviet computer hobby movement of the 1980s. *IEEE Annals of the History of Computing*, 37(1):12–23. doi: 10.1109/MAHC.2015.11. Cited on pag. 66

Stahl and Martensson(2021) Daniel Stahl and Torvald Martensson. Mob programming: From avant-garde experimentation to established practice. *Journal of Systems and Software*, 180: 111017. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2021.111017. URL https://www.sciencedirect.com/science/article/pii/S016412122100114X. Cited on pag. 46

Stanley and Mawer(2008) Graham Stanley and Kyle Mawer. Language learners & computer games: From. *TESL-EJ*, 11(4):n4. Cited on pag. 66

Stephenson(2023) Gareth A Stephenson. Mob programming: The impact on team communication and learning. *School of Electrical and Information Engineering, University of the Witwatersrand.* Cited on pag. 42

Stern and Porr(2017) Phyllis Noerager Stern and Caroline Jane Porr. *Essentials of accessible grounded theory.* Routledge. Cited on pag. 86

Stol *et al.***(2016)** K. J. Stol, P. Ralph and B. Fitzgerald. Grounded theory in software engineering research: A critical review and guidelines. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 120–131. doi: 10.1145/2884781.2884833. Cited on pag. 59, 71, 84, 90

Strauss *et al.***(1998)** A. Strauss, J.M. Corbin and J. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications. ISBN 9780803959408. URL https://books.google.com.br/books?id=wTwYUnHYsmMC. Cited on pag. 71, 73, 84

Strauss(1987) A.L. Strauss. *Qualitative Analysis for Social Scientists*. Cambridge University Press. ISBN 9780521338066. URL https://books.google.com.br/books?id=y16ww5ZsJ0AC. Cited on pag. 71

Strauss and Corbin(1994) Anselm Strauss and Juliet Corbin. Grounded theory methodology: An overview. 273. Cited on pag. 73

Strauss and Corbin(1990) Anselm Strauss and Juliet M Corbin. *Basics of qualitative research: Grounded theory procedures and techniques*. Sage Publications, Inc. Cited on pag. xix, 73, 88

Streiner(2003) David L. Streiner. Starting at the beginning: An introduction to coefficient alpha and internal consistency. *Journal of Personality Assessment*, 80(1):99–103. doi: 10.1207/S15327752JPA8001_18. URL https://doi.org/10.1207/S15327752JPA8001_18. PMID: 12584072. Cited on pag. 80

Stuckey and Swalwell(2014) Helen Stuckey and Melanie Swalwell. Retro-computing community sites and the museum. In *Handbook of Digital Games*, pages 523–547. John Wiley & Sons, Inc. doi: 10.1002/9781118796443.ch19. URL https://doi.org/10.1002/9781118796443.ch19. Cited on pag. 65

Suddaby(2006) Roy Suddaby. From the editors: What grounded theory is not. In *Academy of Management Journal (49:4) 2006*, pages 633–642. Cited on pag. 90

Sutherland and Sutherland(2014) Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time*. Currency. Cited on pag. 11

Švelch(2013) Jaroslav Švelch. Say it with a computer game: Hobby computer culture and the non-entertainment uses of homebrew games in the 1980s czechoslovakia. *Game Studies*, 13(2). Cited on pag. 66

Swamidurai and Umphress(2014) R. Swamidurai and D. Umphress. The impact of static and dynamic pairs on pair programming. In *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, pages 57–63. doi: 10.1109/SERE-C.2014.52. Cited on pag. 30

Thapaliya(2020) Ananga Thapaliya. Evaluation of brain activity while pair programming. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 104–106. Cited on pag. 33

the Wiki(2018) the Wiki. Information about mezuro.org. Wiki of Mezuro Project, 2018. URL http://ccsl.ime.usp.br/wiki/Mezuro. Cited on pag. 6

Tonin(2018) Graziela Simone Tonin. *Technical debt management in the context of agile methods in software development*. Doctoral thesis, Institute of Mathematics and Statistcs, University of Sao Paulo, Brazil. doi: 10.11606/T.45.2018.tde-30072018-142720. URL https://teses.usp.br/teses/disponiveis/45/45134/tde-30072018-142720/. Cited on pag. 107

Tooke and Wynne(2016) Steve Tooke and Matt Wynne. A talk steve tooke and matt wynne gave at qcon london. Video at QCon WebSite, 2016. URL https://www.infoq.com/presentations/distributed-teams-remote-collaboration. Cited on pag. 7

Turnbull(1987) C. Turnbull. *Mountain People*. A Touchstone book. Touchstone. ISBN 9780671640989. URL https://books.google.com.br/books?id=LYH6DKFHNbIC. Cited on pag. 141

Turnbull(2012) C. Turnbull. *The Forest People*. Touchstone. ISBN 9781476728155. URL https://books.google.com.br/books?id=6qzoDB2m2eMC. Cited on pag. 141

Turner(2021) Jonathan Turner. One year of remote mob programming. *Experience Report. XP 2021 Conference. Copyright 2021 is held by the author*. Cited on pag. 45, 46

Umezawa *et al.***(2017)** Katsuyuki Umezawa, Tomohiko Saito, Takashi Ishida, Makoto Nakazawa and Shigeichi Hirasawa. An electroencephalograph-based method for judging the difficulty of a task given to a learner. In *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, pages 384–386. doi: 10.1109/ICALT.2017.18. Cited on pag. 42

Urquhart(2007) Cathy Urquhart. The evolving nature of grounded theory method: The case of the information systems discipline. In *The SAGE Handbook of Grounded Theory*, pages 339–359. SAGE Publications Ltd, 1 Oliver's Yard, 55 City Road, London England EC1Y 1SP United Kingdom. Cited on pag. 61

van Niekerk and Roode(2009) Johanna C. van Niekerk and J. D. Roode. Glaserian and straussian grounded theory: Similar or completely different? In *Proceedings of the 2009 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, SAICSIT '09, pages 96–103, New York, NY, USA. Association for Computing Machinery. ISBN 9781605586434. doi: 10.1145/1632149.1632163. URL https://doi.org/10.1145/1632149.1632163. Cited on pag. 73

Vanhanen and Korpi(2007) J. Vanhanen and H. Korpi. Experiences of using pair programming in an agile project. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 274b–274b. doi: 10.1109/HICSS.2007.218. Cited on pag. 31

Vanhanen *et al.***(2007)** Jari Vanhanen, Casper Lassenius and Mika V. Mantyla. Issues and tactics when adopting pair programming: A longitudinal case study. In *Proceedings of the International Conference on Software Engineering Advances*, ICSEA '07, pages 70–, Washington, DC, USA. IEEE CS. ISBN 0-7695-2937-2. doi: 10.1109/ICSEA.2007.48. URL http://dx.doi.org/10.1109/ICSEA.2007.48. Cited on pag. 2, 26

Vieira and da Silva(2017) Luiz Carlos Vieira and Flavio Soares Correa da Silva. Assessment of fun in interactive systems: A survey. *Cognitive Systems Research*, 41:130–143. ISSN 1389-0417. doi: https://doi.org/10.1016/j.cogsys.2016.09.007. URL https://www.sciencedirect.com/science/article/pii/S1389041716300717. Cited on pag. 39, 160

Vollstedt and Rezat(2019) Maike Vollstedt and Sebastian Rezat. *An Introduction to Grounded Theory with a Special Focus on Axial Coding and the Coding Paradigm*, pages 81–100. Springer International Publishing, Cham. ISBN 978-3-030-15636-7. doi: 10.1007/978-3-030-15636-7_4. URL https://doi.org/10.1007/978-3-030-15636-7_4. Cited on pag. 85

Vosburgh *et al.***(1984)** J. Vosburgh, B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben and Y. Liu. Productivity factors and programming environments. In *Proceedings of the 7th International Conference on Software Engineering*, ICSE '84, page 143?152. IEEE Press. ISBN 0818605286. Cited on pag. 53

Wasiak(2014) Patryk Wasiak. *Playing and Copying: Social Practices of Home Computer Users in Poland during the 1980s*, pages 129–150. Springer London, London. ISBN 978-1-4471-5493-8. doi: 10.1007/978-1-4471-5493-8_6. URL https://doi.org/10.1007/978-1-4471-5493-8_6. Cited on pag. 66

Waters *et al.***(1983)** G. Waters, N. Cutler, N. Reed and L. Watts. *Practice Your BASIC*. Usborne exercise books. Usborne Publishing Ltd. ISBN 9780860207436. URL https://books.google.com.br/books?id=VwPF3sVIOa4C. Cited on pag. xviii, xxiv, 65, 253

Weinberg(1971) Gerald Weinberg. *The Psychology of Computer Programming*. Wiley - Van Nostrand, New York, NY. Cited on pag. 23, 151

Whitworth and Biddle(2007) Elizabeth Whitworth and Robert Biddle. The social nature of agile teams. In *Proceedings of the AGILE 2007*, AGILE 2007, pages 26–36, USA. IEEE Computer

Society. ISBN 0769528724. doi: 10.1109/AGILE.2007.60. URL https://doi.org/10.1109/AGILE.2007.60. Cited on pag. 60

Williams and Kessler(2003) Laurie Williams and Robert Kessler. *Pair programming illuminated*. Pearson Education, Boston, MA. Cited on pag. 1, 2, 24, 25, 30, 31, 151

Williams and Kessler(2000) Laurie Williams and Robert Kessler. The effects of "pair-pressure" and "pair-learning" on software engineering education. In *Proceedings of the 13th Conference on Software Engineering Education & Training*, CSEET '00, pages 59–65, Washington, DC, USA. IEEE Computer Society. ISBN 0-7695-0421-3. URL http://dl.acm.org/citation.cfm?id=794188.794326. Cited on pag. 1, 24, 25, 30

Williams *et al.*(2000) Laurie Williams, Robert R. Kessler, Ward Cunningham and Ron Jeffries. Strengthening the case for pair programming. *IEEE Softw.*, 17(4):19–25. ISSN 0740-7459. doi: 10.1109/52.854064. URL http://dx.doi.org/10.1109/52.854064. Cited on pag. 1, 31

Williams and Stout(2008) W. Williams and M. Stout. Colossal, scattered, and chaotic (planning with a large distributed team). In *Agile 2008 Conference*, pages 356–361. doi: 10.1109/Agile.2008.25. Cited on pag. 28

Wilson(2015) Alexander Wilson. *Mob Programming - What Works, What Doesn't*, pages 319–325. Springer International Publishing, Cham. ISBN 978-3-319-18612-2. doi: 10.1007/978-3-319-18612-2_33. URL https://doi.org/10.1007/978-3-319-18612-2_33. Cited on pag. 38, 39, 51

Wray(2010a) Stuart Wray. Responses to "how pair programming really works". *IEEE Software*, 27(2):8–9. ISSN 0740-7459. doi: 10.1109/MS.2010.51. Cited on pag. 26

Wray(2010b) Stuart Wray. How pair programming really works. *IEEE Softw.*, 27(1):50–55. ISSN 0740-7459. doi: 10.1109/MS.2009.199. URL http://dx.doi.org/10.1109/MS.2009.199. Cited on pag. 26, 27

Wynne(2016) Matt Wynne. Cucumber ltd - the story so far. Post at Cucumber.io WebSite, 2016. URL https://cucumber.io/blog/2016/05/23/cucumber-ltd-story-so-far. Cited on pag. 7

Xu and Correia(2023) Fan Xu and Ana-Paula Correia. Adopting distributed pair programming as an effective team learning activity: a systematic review. *Journal of Computing in Higher Education*. ISSN 1867-1233. doi: 10.1007/s12528-023-09356-3. URL https://doi.org/10.1007/s12528-023-09356-3. Cited on pag. 29

Yuan and Cao(2019) Hans Yuan and Yingjun Cao. Hybrid pair programming - a promising alternative to standard pair programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 1046–1052, New York, NY, USA. Association for Computing Machinery. ISBN 9781450358903. doi: 10.1145/3287324.3287352. URL https://doi.org/10.1145/3287324.3287352. Cited on pag. 29

Zagalo(2013) Nelson Zagalo. Videojogos em portugal: História, tecnologia e arte. *Lisboa: FCA Editora referencial de educação para os media*, 43. Cited on pag. 66

Zarif(2012) Tayyaba Zarif. Interdisciplinary journal of contemporary research in business grounded theory method: An overview. Cited on pag. 71

Zieris and Prechelt(2016) F. Zieris and L. Prechelt. Observations on knowledge transfer of professional software developers during pair programming. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 242–250. Cited on pag. 33

Zieris and Prechelt(2021) Franz Zieris and Lutz Prechelt. Two elements of pair programming skill. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 51–55. doi: 10.1109/ICSE-NIER52604.2021.00019. Cited on pag. 27

Zuill(2014) Woody Zuill. Mob programming: A whole team approach. In *Experience report*, Agile '14. Cited on pag. 23, 37, 38, 39, 61

Zuill(2012) Woody Zuill. YouTube video, 2012. URL https://www.youtube.com/watch?v=p_pvslS4gEI. Cited on pag. 80

Zuill and Meadows(2016) Woody Zuill and Kevin Meadows. *Mob Programming. A Whole Team Approach*. LeanPub, this book is 95% complete - last updated on 2016-10-29 edition. Cited on pag. xvii, xviii, 38, 39, 40, 42, 51

Index

An Ontology of Mob Programming Practices
  Driver rotation techniques, 116
  Practices related to the COVID-19 Pandemic, 122
  Structure breaks, 118
Authorizations of the Ethics Committees, 203
  Plataforma Brasil of the National Commission of Ethics in Research cover sheet, 206
  Ethics Committee of the IME-USP, 203
  Solicitation for the IME-USP Research Committee, 204
  Written Approval of the *Plataforma Brasil* of the National Commission of Ethics in Research, 207–210

Background of the Researcher, 219
  Author, 244
  Certificates of Courses, 226
  Delphi, 243
  Diploma of Honor to Merit, 254
  Institute for Technological Research, 223
  Institute of Mathematics and Statistics, 225
  Microsoft, 240
  Paula Souza State Center for Technological Education, 219
  TK95, 248
  ZX Spectrum, 248
Brazilian Aeronautics Computing Center, 9, 211
Brazilian Air Force, 9, 211

CCA-SJ, 9, 211
Challenges to Start Adopting Mob Programming
  Convincing managers and co-workers, 135
  Different skill levels, 136
  Embracing the Vulnerabilities with Kindness, 139
  Embracing Vulnerabilities with Kindness Theory, 141
  Intense collaboration can be hard, 138
  Interruptions by meetings, 139
  Keeping the Focus, 136
  Mental Exhaustion, 137
  No time for yourself, 140
Claimed Contributions, 14
Classical School Grounded Theory life cycle, 76
CoteiBem, 9, 216, 217
Cucumber.io, 7

Data Analysis techniques inside Grounded Theory, 84
Discussion, 171

Embracing Vulnerabilities with Kindness Theory, 141

Findings
  Related to Managing the Complexity, 145
  Results About Remote Work, 155

Grounded Theory, 70
  Glossary, 69
    Part One, 69
  Background of the Researcher, 219
  Classical School Grounded Theory life cycle, 76
  Data Analysis techniques inside Grounded Theory, 84
  Glossary, 69
    Part Two, 69
  Observations, 63
  Role of the Researcher, 65

Hashload, 8
Hunter Industries, 12, 213

Inforamtion Flow, 107
Information Flow
  Number of participating agents, 56
  Patterns of Interaction, 55
Informed Consent Document
  in English, 199
  in Portuguese, 21, 195
Institution Official Consent Terms, 211
  Summary of Digital Approval of the CoteiBem, 217
  Written Approval of the Brazilian Air Force, 211
  Written Approval of the CoteiBem, 216

Written Approval of the Hunter Industries, 213
Written Approval of the Mob Mentality, 214
Written Approval of the Rocket.Chat, 211
Written Approval of the Shift, 215
Written Approval of the SindicoNet, 218
Interview Script, 191
Interview Script in English, 21, 193
Interview Script in Portuguese, 21, 191

Literature Review
Complexity, 52
Mob Programming, 37
Pair Programming, 24

Methodology, *see* Research Design
Mezuro, 6
Mob Mentality Show, 13, 214
Mob Programming Practices
Driver rotation techniques, 116
Practices related to the COVID-19 Pandemic, 122
Structure breaks, 118

Observations, 63
Ontologies
An Ontology of Mob Programmers' Behaviors, 113
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, 129
An Ontology of Mob Programming Practices, 113
Driver rotation techniques, 116
Practices related to the COVID-19 Pandemic, 122
Structure breaks, 118
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, 129
An Ontology of Mob Programming' Communication Space, 113
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, 129

Practices related to the COVID-19 Pandemic, 122
Administrative Controls, 123
Engineering Controls, 123
Implement Workplace Controls, 123
Personal Protective Equipment (PPE), 124
Safe Work Practices, 124
Problem Outline, 2

Questionnaires, 179
Questionnaire in English, 21, 185
Questionnaire in Portuguese, 21, 179

Research Design, 59
Background of the Researcher, 219
Author, 244
Certificates of Courses, 226
Delphi, 243
Diploma of Honor to Merit, 254
Institute for Technological Research, 223
Institute of Mathematics and Statistics, 225
Microsoft, 240
Paula Souza State Center for Technological Education, 219
TK95, 248
ZX Spectrum, 248
Classical School Grounded Theory life cycle, 76
Context, 4
Brazilian Aeronautics Computing Center, 9, 211
Cotei Bem, 9
Cucumber.io, 7
Hashload, 8
Hunter Industries, 12
Mezuro, 6
Mob Mentality, 13
Rocket.Chat, 7
Shift, 9
SindicoNet, 9
University of Sao Paulo, 5
Data Analysis techniques inside Grounded Theory, 84
Grounded Theory
Background of the Researcher, 219
Data Analysis techniques inside Grounded Theory, 84
Levels of data abstraction, 64
Observations, 63
Research Area, 77
Grounded Theory Glossary, 69
Part One, 69
Part Two, 69
Literature Review, 23
Research Methods
Grounded Theory, 70
Literature Review, 60
Research Area, 77

Research Perspectives, 63
Role of the Researcher, 65
Strategy, 4
Survey, 80
Research Method
Grounded Theory
Data Collection, 78
Definitions of Codes and Coding , 85
Evaluating a Grounded Theory, 89
Major Literature Review, 87
Memoing, 86
Minor Literature Review, 78
Research Area, 77
Role of the Researcher, 65
Sorting, 87
Theoretical Coding, 88
Research Methods
Grounded Theory, 70
Classical School Grounded Theory life cycle, 76
Literature Review, 60
Results
A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors, 161
Challenges to Start Adopting Mob Programming, 135
Convincing managers and co-workers, 135
Different skill levels, 136
Embracing the Vulnerabilities with Kindness, 139
Embracing Vulnerabilities with Kindness Theory, 141
Intense collaboration can be hard, 138
Interruptions by meetings, 139
Keeping the Focus, 136
Mental Exhaustion, 137
No time for yourself, 140
Inforamtion Flow, 107
Mob Programmers' behaviors, 116, 118, 122
Mob Programming Communication Space, 114
Mob Programming Team Roles Ontology, 93
Mob Programming team roles ontology
CyberSecurity Analyst, 106
Driver or Pilot, 95
Facilitator, 100
Navigator or Co-Pilot, 97
Nose, 105
Quality Analyst, 104
Researcher, 102
Well-Defined Team Roles Theory, 108
The First Letter of Saint Paul to the Corinthians, 255
Theory of Benefits of Remote Mob Programming, 163
Theory of Information Flow and Knowledge Sharing about Software Quality Building during Programming, 151
Results Also, *see* Findings
Rocket.Chat, 7, 211
Role of the Researcher, 65

Shift, 9, 215
SindicoNet, 9, 216–218

TCLE, 21, 195
Team Roles, 93
CyberSecurity Analyst, 106
Driver or Pilot, 95
Facilitator, 100
Navigator or Co-Pilot, 97
Nose, 105
Quality Analyst, 104
Researcher, 102
Well-Defined Team Roles Theory, 108
Termo de Consentimento Livre e Esclarecido, 21, 195
The First Letter of Saint Paul to the Corinthians, 255
Theories
A theory about that Mob Programming raises Team Performance, and Remote Mob Programming has an additional advantage. Both approaches, Mob Programming and Remote Mob Programming, contribute positively to the productive factors, including time-to-market reduction. However, this leverage depends on the Complexity, Group Size, Team Roles, Innovative Communication Space, use of Lists, use of CheckLists, use of Maps, and Team Members' Experience and Behaviors, 161
Embracing Vulnerabilities with Kindness Theory, 141

Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, 129
Theory of Benefits of Remote Mob Programming, 163
Theory of Information Flow and Knowledge about Software Quality Building during Programming, 151
Well-Defined Team Roles Theory, 108
Theory of Altruism on Software Development Practices Patterns about How Many Programmers should Implement a Task: Group Size variable, 129
Theory of Benefits of Remote Mob Programming, 163
Theory of Information Flow and Knowledge about Software Quality Building during Programming, 151
Thesis Structure, 19
Part I - Theoretical, 19
Part II - Empirical, 20
Part III - Documents, 20

University of Sao Paulo, 5

Well-Defined Team Roles Theory, 108