Qualidade de software no desenvolvimento com
métodos ágeis

Bruno Henrique Oliveira

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 24 de junho de 2014

Assinatura:

Qualidade de software no desenvolvimento com métodos ágeis

Bruno Henrique Oliveira

Orientadora: Profa. Dra. Simone do Rocio Senger de Souza

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Junho de 2014

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

048q Oliveira, Bruno Henrique
Qualidade de software no desenvolvimento com
métodos ágeis / Bruno Henrique Oliveira;
orientadora Simone do Rocio Senger de Souza. -- São
Carlos, 2014.
85 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2014.

1. Qualidade de Software. 2. Métodos Ágeis. 3.
Qualidade Ágil. 4. Cultura Ágil. I. Souza, Simone
do Rocio Senger de, orient. II. Título.

Agradecimentos

A GRADEÇO à Prof^a. Dr^a. Simone do Rocio Senger de Souza pela amizade, conselhos e capacidade de direcionamento do projeto nos momentos mais delicados. Esses fatores essenciais para o desenvolvimento do trabalho. Aos meus pais Marina e Guilherme, pela compreensão e incentivo a distância durante essa caminhada. Ao Instituto de Ciências Matemáticas e de Computação pela oportunidade e ambiente de pesquisa excelentes. Aos funcionários e professores do ICMC, em especial o Prof^o. Dr. João Porto de Albuquerque por sua importante contribuição nos trabalhos empíricos deste projeto de mestrado. Ao CNPq pelo apoio financeiro. Às empresas e profissionais que participaram dos estudos empíricos deste trabalho, pela colaboração, tempo e paciência. Agradeço também ao Prof^o. Dr. Flavio Moreira de Oliveira, aos amigos Elder, Maicon, Leandro e Artur e toda a equipe do Centro de Pesquisa em Engenharia de Software (CePES) da PUC-RS pelo auxílio nos estudos realizados no Rio Grande do Sul e por terem me recebido tão bem durante minha estadia em Porto Alegre. Aos amigos Ricardo Fontão Verhaeg, pelo auxílio na construção da ferramenta IQA, e Filipe Del Nero Grillo que vivenciou a mesma experiência do mestrado durante esses anos. Agradeço especialmente a minha namorada Juliana Mariotti Guerra que me apoiou imensamente durante todo o projeto. Não poderia deixar de destacar o quanto o apoio da Juliana foi importante nesses anos de trabalho.

A cura para o tdio a curiosidade.
No existe cura para a curiosidade.

Dorothy Parker

A Engenharia de Software é uma disciplina que tem entre seus objetivos melhorar a produtividade dos processos de desenvolvimento de software, assim como propiciar qualidade ao produto resultante desses processos. Para mensurar a qualidade dos produtos de software, foram criados modelos de qualidade, que recomendam métricas, processos e atividades que passaram a se tornar parte do dia-a-dia do desenvolvimento de projetos em empresas. Considerando outra perspectiva, a indústria de software tem adotado cada vez mais os métodos ágeis. Esses métodos foram desenvolvidos visando a entrega rápida do software, com ciclos curtos e adaptáveis de desenvolvimento, foco na comunicação direta e baixo volume de documentação. Considerando a importância do tema qualidade de software, e a baixa aderência dos modelos tradicionais de qualidade aos métodos ágeis, o objetivo deste projeto foi investigar o tema qualidade de software no contexto ágil, ou seja, estudar quais métricas de qualidade são empregadas nesse processo de desenvolvimento. Para isso foram realizados dois estudos empíricos, um estudo de caso e um *survey*, sobre atividades de garantia e controle de qualidade, métricas de qualidade de software, processos e ferramentas utilizadas no desenvolvimento de software. Os resultados obtidos guiaram a construção de uma ferramenta de apoio para avaliação da qualidade durante o desenvolvimento ágil de software. Os resultados dos estudos mostraram que a execução constante de atividades como revisão de código e refatoração, são fatores essenciais para garantia de qualidade nos métodos ágeis. Outro resultado encontrado foi o de que praticantes de métodos ágeis são entusiastas do processo de desenvolvimento utilizado. Eles conhecem o método e praticam com alta fidelidade os passos definidos pelo processo. É possível concluir que os métodos ágeis possuem diversas atividades como foco na garantia de qualidade de seu produto desde os estágios iniciais do desenvolvimento. A cultura ágil cria um ambiente propício para motivação e engajamento das equipes de desenvolvimento, fato que reflete positivamente na qualidade final dos produtos.

Abstract

One of the main objectives of Software Engineering is to improve the productivity of software development processes, as well as providing quality to the product resulting from such processes. Thus, quality models were defined to measure the software quality. Those models recommend metrics, processes and activities that became part of day-to-day on development companies. Considering another perspective, the software industry has increasingly adopted agile methods. These methods were developed considering rapid software delivery, with short and adaptable development cycles, focusing on direct communication and low volume of documentation. Considering the importance of software quality and the low compliance of agile methods to traditional quality models, this project aimed to investigate software quality in agile development environments, in other words, to research which quality metrics are employed in these development processes. Considering this objective, two empirical studies were designed, a case study and a survey. These studies have explored themes like software quality control, software quality assurance, quality metrics, development process and development tools that are employed on software development. The results guided the construction of a tool to support the quality evaluation during the agile development. The studies' results showed that the high frequency of activities such as code review and refactoring, are essential factors for assuring quality on projects using agile methods. Another result was found regarding developer's behavior. Agile practitioners are enthusiasts of the development process they use. They have a high level of compliance to development process they use. It is possible to conclude that agile methods have several activities focused on the quality assurance of its own products since the initial stages of development. The agile culture creates a convenient environment that engages and motivates the development teams. This fact has a positive effect on the product quality.

Sumário

1	Introdução	1
1.1	Contextualização	1
1.2	Motivação	3
1.3	Objetivo	3
1.4	Metodologia de Pesquisa	4
1.5	Organização desta Monografia	5
2	Fundamentação Teórica	7
2.1	Considerações Iniciais	7
2.2	Qualidade de Software	8
2.3	Modelos de Qualidade de Software	9
2.4	Controle e Garantia de Qualidade	14
2.4.1	Revisões	15
2.4.2	Refatoração	16
2.4.3	Teste de Software	17
2.5	Métricas de Qualidade	18
2.6	Métodos Ágeis	22
2.7	Métodos Ágeis Utilizados na Pesquisa	25
2.7.1	SCRUM	25
2.7.1.1	Práticas	26
2.7.1.2	Processo	27
2.7.1.3	Papéis e responsabilidades	28
2.7.2	Extreme Programming	29
2.7.2.1	Práticas	31
2.7.2.2	Processo	33
2.7.2.3	Papéis e responsabilidades	34
3	Estudo de Caso - Qualidade de Software	37
3.1	Considerações Iniciais	37
3.2	Objetivo	38
3.3	Protocolo	38
3.3.1	Estratégia de Seleção	38
3.3.2	Triangulação Metodológica	39

3.3.3	Entrevistas	39
3.3.4	Ameaças à Validade	40
3.3.5	Piloto	42
3.3.6	Questões	43
3.4	Coleta de Dados	43
3.4.1	Empresa MA1	44
3.4.2	Empresa MA2	45
3.4.3	Empresa MT1	46
3.4.4	Empresa MT2	47
3.5	Análise de Dados	48
3.6	Dificuldades Encontradas	50
3.7	Considerações Finais	51
4	Survey - Qualidade e Cultura Ágil	53
4.1	Considerações Iniciais	53
4.2	Objetivo	54
4.3	Protocolo	54
4.3.1	Questões	55
4.3.2	Ameaças a Validade	55
4.4	Coleta de Dados	57
4.4.1	Análise dos Dados Quantitativos	57
4.4.2	Análise dos Resultados Qualitativos	60
4.5	Considerações Finais	61
5	IQA - Uma ferramenta de avaliação de qualidade	63
5.1	Considerações Iniciais	63
5.2	Desenvolvimento da Aplicação	64
5.2.1	Implementação	64
5.2.2	Questões	65
5.3	Uso da Ferramenta em um Ambiente Real	67
5.4	Coleta de Dados	67
5.5	Análise de Dados	68
5.6	Considerações Finais	70
6	Conclusão	71
6.1	Caracterização da Pesquisa Realizada	71
6.2	Fatores que contribuem para qualidade do produto	72
6.3	Contribuições	73
6.4	Dificuldades e Limitações	74
6.5	Trabalhos Futuros	75
	Referências	82
A	Questões - Estudo de Caso	83
A.1	Questões sobre o método de desenvolvimento	83
A.2	Questões sobre qualidade de software	84

Lista de Figuras

1.1	Representação simplificada das quatro fases da pesquisa ação	4
2.1	Modelo de qualidade de McCall	10
2.2	Modelo de qualidade de Boehm	12
2.3	Relações entre critérios Perry, 1987)	12
2.4	Modelo de qualidade interna e externa da ISO/IEC 9126	13
2.5	Estrutura do modelo hierárquico GQM (Basili et al., 1994)	20
2.6	Processo de Desenvolvimento SCRUM (Abrahamsson et al., 2002)	27
2.7	Ciclo de Vida do Processo XP (Abrahamsson et al., 2002)	33
3.1	Kanban utilizado na empresa MA1	44
3.2	Atividades de garantia de qualidade identificadas nas empresas.	49
4.1	Distribuição da utilização dos métodos ágeis entre os participantes do <i>survey</i>	57
4.2	Classificação da descrição dos métodos utilizados pelos participantes em comparação com a literatura.	58
4.3	Atividades realizadas pelos participantes a cada iteração de desenvolvimento	58
4.4	Distribuição dos participantes de acordo com participação em equipes de desenvolvimento que possuem equipe dedicada a atividade de testes.	59
4.5	Utilização de ferramentas de apoio a atividade de teste de software pelos participantes.	59
4.6	Forma com que as equipes dos participantes geram os casos de teste para cada iteração.	60
5.1	Padrão de Desenvolvimento MVC (Model-View-Controller)	65
5.2	Tela da Ferramenta IQA	66
5.3	Gráficos gerados pela ferramenta IQA. A Figura 5.3a mostra a distribuição de opinião dos desenvolvedores sobre qual o foco da iteração realizada. A Figura 5.3b mostra a visão do cliente sobre o foco da mesma iteração.	69
5.4	Gráfico informativo gerado pela IQA sobre frequência de utilização de ca- nais de comunicação.	69

Lista de Tabelas

2.1	Critérios de Qualidade de McCall	11
2.2	Tipos de revisão de qualidade	16
2.3	Métricas por fator de qualidade (Humphrey, 1987)	19
3.1	Validade de Construção	41
3.2	Validade Externa	42
3.3	Validade de Conclusão	42
3.4	Empresas selecionadas para o estudo de caso sobre qualidade de software. .	43
3.5	Análise SWOT de qualidade para métodos tradicionais	50
3.6	Análise SWOT de qualidade para métodos ágeis	51
4.1	Questões do <i>survey</i>	56

Lista de Abreviaturas e Siglas

ASD	<i>Desenvolvimento adaptável de software (do inglês Adaptive Software Development)</i>
CMM	<i>Modelo de capacidade da maturidade (do inglês Capability Maturity Model)</i>
CMMI	<i>Integração do modelo de capacidade da maturidade (do inglês Capability Maturity Model Integration)</i>
CRM	<i>Sistema de gestão de relacionamento com clientes (do inglês Customer Relationship Management)</i>
CRYSTAL	<i>Processo de desenvolvimento ágil de software Crystal Clear</i>
DRE	<i>Eficiência de remoção de defeitos (do inglês Defect Removal Efficiency)</i>
ERP	<i>Sistemas Integrados de Gestão Empresarial (do inglês Enterprise Resource Planning)</i>
GQM	<i>Objetivo/Questão/Métrica (do inglês Goal Question Metric)</i>
HTML	<i>Linguagem de marcação de hipertexto (do inglês HyperText Markup Language)</i>
IEC	<i>Comitê internacional Eletrotécnico (do inglês International Electrotechnical Commission)</i>
IEEE	<i>Instituto de engenheiros eletricitistas e eletrônicos (do inglês Institute of Electrical and Electronics Engineers)</i>
IQA	<i>Ferramenta de avaliação de qualidade de iterações (do inglês Iteration Quality Assessment)</i>
ISO	<i>Organização internacional de padronização (do inglês International Organization for Standardization)</i>
KLOC	<i>Mil linhas de código (do inglês Kilo Lines of Code)</i>
MTTC	<i>Tempo médio para se realizar uma mudança (do inglês Mean Time To Change)</i>
MVC	<i>Padrão de desenvolvimento modelo-apresentação-controlador (do inglês Model-View-Controller)</i>
NBR	<i>Denominação de norma da Associação Brasileira de Normas Técnicas (ABNT)</i>

PHP	<i>Linguagem de programação. PHP significa pré-processador de hipertexto (do inglês PHP Hypertext Preprocessor)</i>
QA	<i>Garantia de qualidade (do inglês Quality Assurance)</i>
QC	<i>Controle de qualidade (do inglês Quality Control)</i>
RAD	<i>Método de desenvolvimento rápido de software (do inglês Rapid Application Development)</i>
RIPP	<i>Método de desenvolvimento ágil (do inglês Rapid Iterative Production Prototyping)</i>
RUP	<i>Processo de desenvolvimento de software (do inglês Rapid Unified Process)</i>
SCRUM	<i>Processo de desenvolvimento ágil de software</i>
SEI	<i>Instituto de engenharia de software (do inglês Software Engineering Institute)</i>
SPICE	<i>Norma que define um processo de desenvolvimento de software (do inglês Software Process Improvement and Capability Determination)</i>
SWEBOK	<i>Guia de referência para engenharia de software (do inglês Software Engineering Body of Knowledge)</i>
SWOT	<i>Método utilizado para identificação de força, fraqueza, oportunidades e ameaças muito utilizada para análise de projetos (do inglês Strengths, Weaknesses, Opportunities e Threats)</i>
TDD	<i>Desenvolvimento orientado a testes (do inglês Test Driven Development)</i>
XP	<i>Método de desenvolvimento ágil Extreme Programming</i>

Introdução

1.1 Contextualização

A Engenharia de Software é uma disciplina que visa proporcionar produtividade ao processo de desenvolvimento de software, assim como propiciar qualidade ao produto resultante desse processo. Sobretudo, objetiva-se a redução dos custos e esforços geralmente exigidos nas várias fases do desenvolvimento e da manutenção do software (Pressman, 2010). Nesta direção, metodologias convergem e contribuem para reutilização de técnicas, mecanismos e experiências bem sucedidas.

Diferentes processos de desenvolvimento de software vêm sendo definidos e adotados pela indústria (Pressman, 2010). De uma maneira geral, esses processos definem etapas repetíveis para o desenvolvimento do software, utilizando muitas vezes técnicas para o gerenciamento e controle do projeto. A escolha do processo é influenciada pelo domínio das aplicações desenvolvidas e pelas características das empresas, não existindo um modelo de processo padrão para todas as aplicações e tipos de negócios (Sommerville, 2004).

Nesse sentido, observa-se que a utilização de métodos ágeis para o desenvolvimento de software vem despertando o interesse, tanto da academia, como da indústria (Boehm, 2002a). Esses métodos foram desenvolvidos visando a entrega rápida do software, focando na contínua interação entre desenvolvedores e clientes, na tentativa de aumentar a probabilidade de que um software atenderá às necessidades e às mudanças de requisitos dos clientes. Prazos de entregas curtos, associados a complexidade dos processos de

desenvolvimento existentes, foram a principal motivação para que se definissem métodos ágeis (Cockburn et al., 2001; Fowler, 2005).

O desenvolvimento de software de alta qualidade é um tema de grande e crescente importância na indústria de software (Gillies, 2002a). Existem diversas visões para qualidade de software, que focam em diferentes aspectos. Na literatura verifica-se além das diversas definições de qualidade, que qualidade pode ter significados diferentes para pessoas diferentes, tais como usuário, clientes e gerentes. No livro *Software Quality: Theory and Management* o autor afirma que “qualidade é transparente quando presente, porém facilmente percebida em sua ausência” (Gillies, 2002b). Juran e Feo (2010) define qualidade como “conformidade para uso” enquanto Crosby (1992) afirma que qualidade é “conformidade aos requisitos”.

Consolidando-se essas visões, podemos dizer que a qualidade de software pode ser definida como (IEEE, 1991):

- O nível com o qual cada sistema, componente ou processo atende às suas especificações;
- O nível com o qual cada sistema, componente ou processo atende às necessidades ou expectativas dos clientes.

Visando mensurar a qualidade dos processos de software, foram criados modelos de qualidade, que recomendam processos, atividades e métricas para aferir a qualidade de um software. O CMM - *Capability Maturity Model* é provavelmente o modelo de qualidade mais conhecido. Ele foi desenvolvido pelo SEI - *Software Engineering Institute*, Universidade Carnegie Mellon e financiado pelo Departamento de Defesa Norte-Americano, com o objetivo de estabelecer um padrão de qualidade para o desenvolvimento de software. O CMM, o qual evoluiu para o CMMI - *Capability Maturity Model Integration*, é um modelo que se baseia em um processo gradual, que leva as organizações a se aprimorarem continuamente, na busca de soluções próprias para os problemas existentes no desenvolvimento de software (Rocha et al., 2001). O modelo apresenta um caminho evolucionário de melhoria de maturidade, o que ocorre por meio do atendimento de práticas-chave distribuídas em cinco níveis de maturidade ou através de um modelo contínuo quando o foco de melhoria é restrito a algumas áreas do desenvolvimento. Quanto maior o nível, maior é a maturidade dos processos de desenvolvimento de software em uma organização.

Atividades como as de garantia de qualidade e métricas como confiabilidade, eficiência, segurança e manutenibilidade passaram a se tornar parte do dia-a-dia do desenvolvimento de um projeto em grandes empresas (Carosia, 2004). A frequente avaliação de qualidade de produtos e processos é denominado Garantia de Qualidade (*Quality Assurance*). Garantia de Qualidade é definida como uma estrutura composta por responsabilidades,

procedimentos, processos e recursos para implementar a gestão da qualidade no produto e/ou processo (Pressman, 2010). Os sistemas de Garantia de Qualidade buscam auxiliar as organizações a garantir a satisfação de seus clientes através de seus produtos e serviços, alcançando a expectativa dos mesmos (Chulani et al., 2003; McCall, 1979; Pressman, 2010). Esses sistemas cobrem uma vasta gama de atividades a serem realizadas durante todo o ciclo de vida de um produto, passando por atividades no planejamento, desenvolvimento, manutenção e suporte (Pressman, 2010), e portanto, durante todo o processo de software.

1.2 Motivação

Apesar da importância dos métodos ágeis e do reconhecimento de sua contribuição positiva para a qualidade do produto e satisfação do cliente, observa-se que os modelos de garantia de qualidade existentes não aderem facilmente a esses métodos, dado a agilidade imposta pelos mesmos. Existem poucos trabalhos publicados sobre o assunto qualidade de software em produtos desenvolvidos com métodos ágeis.

Considerando essa afirmação, surgem algumas dúvidas sobre a garantia de qualidade em métodos ágeis, como: Qual o impacto do desenvolvimento ágil de software na qualidade dos produtos finais? Quais fatores contribuem para melhoria da qualidade de um produto desenvolvido com métodos ágeis? Esta última pergunta é de extrema importância pois guia a condução deste projeto de mestrado.

Uma revisão de literatura mostra que os modelos de qualidade de software existentes foram construídos para abordagens tradicionais de software, que em grande parte derivam de modelos sequenciais e em cascata (Abbas, 2009). Dessa forma é difícil utilizar as técnicas e processos existentes de garantia de qualidade no desenvolvimento ágil. Foram realizados alguns estudos empíricos sobre qualidade de software com métodos ágeis, principalmente com foco no SCRUM e/ou XP, porém as iniciativas são muito dispersas. Dyba e Dingsoyr conduziram uma revisão sistemática sobre estudos empíricos em desenvolvimento de software ágil. Nessa revisão os autores concluem que apesar de muitos trabalhos sobre o método ágil XP, ainda existe a necessidade de mais e melhores estudos empíricos sobre métodos ágeis (Dyba e Dingsoyr, 2008).

1.3 Objetivo

Considerando a grande importância do assunto qualidade de software e a baixa aderência dos modelos atuais de qualidade aos métodos ágeis, cada vez mais utilizados na indústria

e na pesquisa, o objetivo deste projeto é identificar os fatores que contribuem para a qualidade de software no desenvolvimento ágil.

Para isso foram pesquisados trabalhos relacionados à atividades de garantia de qualidade, métricas, processos e ferramentas utilizadas no desenvolvimento ágil. Foram realizados também estudos empíricos sobre práticas de qualidade empregadas nas empresas que desenvolvem softwares com métodos ágeis.

1.4 Metodologia de Pesquisa

Esse projeto de pesquisa iniciou-se com foco em identificar atividades ferramentas e métricas utilizadas nos métodos ágeis de desenvolvimento que contribuíssem de fato para melhoria ou garantia de qualidade dos produtos de softwares resultantes desses processos de desenvolvimento.

Ao longo do projeto, após a realização de estudos de caso, foi possível perceber que existem outros fatores possivelmente mais relevantes para a garantia e aumento da qualidade de produtos de software desenvolvidos com métodos ágeis.

O método de pesquisa foi então substituído pela pesquisa ação. Pesquisa ação é toda tentativa continuada, sistemática e empiricamente fundamentada de aprimorar a prática (Tripp, 2005). A Figura 1.1 mostra as quatro fases de uma pesquisa ação. Com essa orientação metodológica é possível produzir informações e conhecimentos de uso mais efetivo quando os objetos de estudo não estão totalmente claros, ou quando podem existir fatores externos que afetem a pesquisa.

Figura 1.1: Representação simplificada das quatro fases da pesquisa ação

Um estudo de caso realizado em quatro empresas indicou uma diferença comportamental entre desenvolvedores que utilizam métodos ágeis e desenvolvedores que utilizam

outros métodos. Os desenvolvedores de métodos ágeis se mostraram mais comprometidos com o processo de desenvolvimento e a qualidade do produto.

1.5 Organização desta Monografia

Neste capítulo foi apresentado o contexto no qual este projeto de mestrado se insere e a motivação para desenvolver o trabalho proposto. O restante dessa dissertação está organizada da seguinte forma. No Capítulo 2 são apresentados conceitos relacionados à qualidade de software, modelos de qualidade, controle e garantia de qualidade. Serão apresentados também o conceito de métodos ágeis e uma síntese sobre os principais métodos ágeis existentes. No Capítulo 3 é apresentado um estudo de caso realizado em empresas com objetivo de entender como as mesmas compreendem e enfrentam os desafios de se produzir um software de qualidade. O estudo foi realizado com quatro empresas sendo elas duas praticantes de métodos ágeis e outras duas que praticavam método em cascata. O estudo de caso relatado no Capítulo 3 motivou a criação de um *survey* que é descrito no Capítulo 4. Esse *survey* foi realizado com praticantes de métodos ágeis com intuito de esclarecer a relação de praticantes de métodos ágeis com o processo de desenvolvimento e atividades de controle e garantia de qualidade. No Capítulo 5 é relatada a construção de uma ferramenta web para acompanhamento de iterações ágeis com foco em pontos específicos de qualidade, voltado em um primeiro momento para os desenvolvedores. Por fim, no Capítulo 6 se encontram as conclusões desta pesquisa.

Fundamentação Teórica

2.1 Considerações Iniciais

Grande parte da sociedade, empresas, faculdades, institutos de pesquisas e governos dependem de sistemas de software para realizar suas tarefas do dia-a-dia. Considerando esta importância, a qualidade dos produtos de software passa a ser ponto crucial no desenvolvimento, operação e manutenção desses sistemas (Rocha et al., 2001).

A Engenharia de Software tem entre seus grandes objetivos aumentar a qualidade de software. Para isso são investigados aspectos de qualidade de produtos de software e dos processos de desenvolvimento dos mesmos. Para muitos, a qualidade do processo de software é tão importante quanto a qualidade do produto. Por esse motivo, na década de 90, houve uma grande preocupação com a modelagem e melhoria do processo de software (Pfleeger, 2001; Rocha et al., 2001).

Um dos maiores problemas de qualidade de software é a sua gestão. Poucas pessoas possuem informação ou conhecimento necessário para garantir a qualidade de um produto. Alguns desenvolvedores acreditam que a qualidade é algo pra se preocupar após o produto estar finalizado (Pressman, 2010).

O termo garantia de qualidade de software é um conceito guarda-chuva, composto por atividades que devem ser aplicadas durante todo o processo de desenvolvimento. De fato, segundo Pressman (2010) é necessário um controle do processo de desenvolvimento para que até as tarefas mais simples sejam executadas de maneira padronizada, reduzindo

assim a distância entre um produto e outro e facilitando o controle de variáveis como taxa de defeitos, horas de desenvolvimento e recursos financeiros.

Diante deste cenário foram desenvolvidos diversos modelos para processo de desenvolvimento de software, como por exemplo a ISO 9126. Esses modelos abordam formas de se minimizar a variação entre os produtos de software e sugerem que, melhorando o processo de desenvolvimento, pode-se melhorar a qualidade do produto final (Pfleeger, 2001).

Uma das maiores dificuldades da gestão de qualidade é que sua definição pode ser diferente para cada pessoa envolvida em um projeto. Cada pessoa ou empresa possui a sua visão própria de qualidade. Gillies (2002a), afirma que “qualidade são as pessoas”. Ele explica que as pessoas são quem enfrentam problemas a serem resolvidos por software, as pessoas definem o problema, e as pessoas encontram a solução. Elas utilizarão o sistema e farão seus próprios julgamentos sobre sua qualidade. Além disso, Gillies (2002a) afirma que o termo qualidade é particularmente problemático quando trata o assunto desenvolvimento de software. Isso ocorre por que um software não possui existência física, os clientes não sabem exatamente quais são suas necessidades, os requisitos não são imutáveis, a evolução de hardware e software são extremamente rápidas e a expectativa dos consumidores é sempre alta e pouco gerenciada. Por isso é necessário, a definição de um modelo de qualidade, com métricas tangíveis que reflitam uma forma de avaliar os produtos de software.

Esse capítulo aborda as definições de qualidades e métodos ágeis pesquisadas para contextualização deste trabalho.

2.2 Qualidade de Software

Existem inúmeras definições de qualidade, que se complementam em aspectos diferentes na qualidade do produto e/ou do processo de desenvolvimento.

De acordo com NBR ISO 8402/1994, a qualidade é totalidade das características de uma entidade, que confere a capacidade de satisfazer necessidades implícitas e explícitas da mesma. Considerando o contexto de software, podemos afirmar que a Qualidade de Software é o conjunto de características que devem ser alcançadas para suprir as necessidades de um cliente.

A definição de qualidade de software pode ainda ser dividida em dois níveis; o primeiro considerando a qualidade do produto e atributos como confiabilidade e taxa de defeitos. Kan (2002) denomina o primeiro nível como “q minúsculo”. O segundo nível é mais abrangente e considera a qualidade do produto, do processo e a satisfação do cliente (Kan, 2002). Esse nível é denominado “Q Maiúsculo”.

Apesar da qualidade ser uma característica ou atributo de um determinado objeto e portanto pode ser medida e comparada com padrões previamente definidos, o software é uma entidade conceitual, e portanto mais complexo de se caracterizar do que objetos físicos, dificultando assim a definição da qualidade de um software específico. Entretanto, existem medições de programas de computadores.

Quando um item é avaliado em características mensuráveis, é possível identificar dois tipos de qualidade. A qualidade de projeto e a de conformidade com as especificações. Qualidade de projeto se refere às características que projetistas planejam para um item, como por exemplo, indicadores de tolerância e desempenho. A qualidade de conformidade é o nível com que um produto atende às suas especificações (Pressman, 2010).

É possível perceber que a definição de qualidade pode ser modificada ou estendida. Neste sentido, este trabalho foca em três pontos de qualidade indicados por Pressman (2010)

1. Requisitos de software são a base para se realizar medidas de qualidade. A falta de conformidade com as especificações é um problema de falta de qualidade.
2. Padrões especificados definem um conjunto de critérios de desenvolvimento que guiam a forma como o software será desenvolvido. A não conformidade com as definições desses padrões resultará em falta de qualidade.
3. O software deve obedecer requisitos que muitas vezes estão implícitos, mas são esperados de um software profissional, como manutenibilidade. A ausência desses itens torna a qualidade de um software contestável.

Existem também diversas características que são diretamente afetadas pelo processo de desenvolvimento do software. Essas características podem ser detalhadas em vários níveis, formando um grande conjunto de atributos que descrevem a qualidade de um produto de software (Rocha et al., 2001). Assim para que se possa comparar a qualidade de produtos de software foram criados padrões, os quais são denominados modelos de qualidade. Esses modelos são descritos na próxima seção.

2.3 Modelos de Qualidade de Software

Modelos de qualidade são necessários para se avaliar a qualidade de produtos de software em diferentes situações. Os primeiros trabalhos significativos nessa área são os de McCall (1979) e Boehm et al. (1978). Eles apresentam uma hierarquia de características que contribuem para a qualidade do produto. Muitos outros modelos foram propostos após esses, inclusive a norma ISO 9126, criada em 1991, que surgiu como tentativa de consolidar as diferentes visões de qualidade em uma norma internacional.

A ideia de modelo hierárquico de qualidade é antiga, porém ainda muito utilizada. Os modelos hierárquicos são baseados em um conjunto de critérios de qualidade, cada um associado com uma ou mais métricas. Quando se abordam requisitos de qualidade, devem ser planejados quais critérios devem ser utilizados, como eles se relacionam e quais as métricas associadas a esse critério e como elas podem ser combinadas em uma informação significativa sobre a qualidade (Gillies, 2002a).

O modelo de qualidade apresentado por McCall (1979) tem como foco a sua utilização durante o processo de desenvolvimento. O modelo identifica três áreas do processo de desenvolvimento de um software: Operação de Produto, Revisão de Produto e Transição de Produto. Cada área possui um número de critérios conforme mostrado na Figura 2.1 e cada um desses critérios está descrito na Tabela 2.1. Os critérios nesse modelo foram escolhidos para refletir a visão de desenvolvedores e usuários, na tentativa de aproximar as duas visões.

Figura 2.1: Modelo de qualidade de McCall

Outros modelos alternativos ao de McCall (1979) foram propostos como os de Rocha (1983) e Dromey (1996). O modelo de Evans e Marciniak (1987) não considerava fatores como testabilidade, e por isso não será aprofundado. O modelo de Deutsch e Willis (1988) contém quinze fatores classificados em quatro categorias. Comparando com o modelo de McCall (1979) foram introduzidos cinco novos fatores: verificabilidade, escalabilidade, segurança, sobrevivência e gerenciabilidade.

Tabela 2.1: Critérios de Qualidade de McCall

Fator de qualidade de software	Descrição
Assertividade	Medida que afere o nível com que um software satisfaz suas especificações
Confiabilidade	Medida que afere se o sistema realiza suas atividades sem falhas
Eficiência	Quantidade de recursos computacionais utilizados por um sistema para executar suas atividades
Integridade	Medida que indica se os dados estão coerentes e precisos ao longo do sistema
Usabilidade	A facilidade em se utilizar o software
Manutenibilidade	O esforço necessário para se encontrar e corrigir um problema após o software estar em uso
Flexibilidade	O esforço necessário para se modificar o sistema
Testabilidade	A facilidade em se testar o programa e garantir que ele está livre de erros
Portabilidade	O esforço necessário para portar o sistema de uma configuração de hardware ou software para outra
Reusabilidade	A facilidade de se reutilizar o software em diferentes contextos
Interoperabilidade	O esforço necessário para integrar o sistema com outros sistemas

Boehm et al. (1978) propuseram um modelo hierárquico que engloba os critérios presentes no modelo de McCall (1979), estendendo-o de modo a acrescentar novos critérios de qualidade. A Figura 2.2 mostra os três níveis da hierarquia do modelo de Boehm et al. (1978). O nível mais alto da estrutura reflete as utilizações feitas pelo sistema. O nível mais baixo da estrutura fornece uma série de características que quando combinadas formam o nível intermediário de características que definem as condições necessárias para se atingir qualidade (Boehm et al., 1978).

Apesar dos modelos de McCall (1979) e Boehm et al. (1978) especificarem seus modelos de qualidade até o nível de métricas, indicadores individuais de qualidade de software não fornecem uma visão geral da qualidade do produto e portanto esses indicadores devem ser combinados. Perry (1987) propôs um relacionamento entre esses indicadores, como mostra a Figura 2.3. Gillies (2002a) criticou muito o trabalho realizado por Perry (1987) pois ele assume que os relacionamentos entre os indicadores são comutativos, o que nem sempre é verdadeiro.

A ISO (*International Organization for Standardization*) e o IEC (*International Electrotechnical Commission*), baseados em tentativas anteriores de se definir qualidade de software, publicaram o padrão ISO/IEC 9126 que define um modelo de qualidade para produtos de software, características de qualidade de software e métricas relacionadas.

Figura 2.2: Modelo de qualidade de Boehm

Correção	C									
Confiabilidade	-	R								
Eficiência			E							
Integridade			•	I						
Usabilidade	-	-	•	-	U					
Manutenibilidade	-	-	•		-	M				
Testabilidade	-	-	•		-	-	T			
Flexibilidade	-	-	•		-	-	-	F		
Portabilidade			•			-	-		P	
Reusabilidade		•	•	•		-	-	-	-	R
Interoperabilidade			•	•					-	I

{•} = Inverso {vazio} = Neutro {-} = Direto

Figura 2.3: Relações entre critérios (Perry, 1987)

Essas definições nos permitem avaliar e criar metas para a qualidade de um produto de software.

O modelo de qualidade da ISO/IEC 9126 possui duas partes. A primeira parte do modelo é aplicável para modelagem da qualidade interna e externa de um produto de software, enquanto a segunda parte tem o objetivo de modelar a qualidade em uso de um produto de software. As duas partes são necessárias para que possamos modelar a qualidade de um produto de software em fases diferentes do seu ciclo de desenvolvimento.

Tipicamente, qualidade interna é obtida através de revisões de documentos de especificação, verificação de modelos ou por análise estática do código-fonte. A qualidade externa refere-se às propriedades de software que interagem com o ambiente.

Por outro lado, a qualidade em uso refere-se a qualidade percebida pelo usuário final. Essas qualidades de produto em diferentes estágios do desenvolvimento não são isoladas e são influenciadas uma pela outra. Portanto, métricas internas podem ser utilizadas para prever a qualidade final de um produto de software ainda no início do seu desenvolvimento.

Para modelar qualidade interna e externa, a ISO/IEC 9126 define o mesmo modelo. Essa versão genérica pode ser instanciada como modelo para qualidade interna e externa através da utilização de um conjunto diferente de métricas. O modelo é baseado nas seis características: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade. A Figura 2.4 mostra essas características e suas sub-características.

Figura 2.4: Modelo de qualidade interna e externa da ISO/IEC 9126

O modelo de qualidade em uso é baseado nas características: efetividade, produtividade, segurança, satisfação e não possui nenhuma sub-característica. São definidas ao longo da ISO/IEC 9126 métricas para aferir as sub-características citadas anteriormente. Essas métricas são abstratas tornando-as aplicáveis para vários tipos de produtos de software, entretanto é necessário que seja feito um refinamento de acordo com o tipo da aplicação.

2.4 Controle e Garantia de Qualidade

Controle de qualidade (QC¹) abrange uma série de inspeções, revisões e testes utilizados durante o processo de desenvolvimento de software, na tentativa de garantir que o produto final cumpra com os requisitos especificados. O QC é um mecanismo de feedback para o processo de desenvolvimento de software. A combinação de medições e feedbacks trazem informações que podem auxiliar na melhoria dos processos de desenvolvimento quando necessário. As atividades de QC podem ser totalmente automatizadas, manuais, ou uma combinação das duas. O conceito chave do QC é que todos os produtos devem possuir especificações definidas e mensuráveis e portanto que podem ser comparadas com a saída de cada atividade do processo de desenvolvimento.

Garantia de qualidade (QA²) é um termo mais abrangente do que QC e consiste em abastecer a gestão de um projeto com dados sobre a qualidade de um produto. De acordo com a IEEE, QA é um conjunto de ações necessárias para alcançar qualidade de software, incluindo qualidade do produto de software e do processo de desenvolvimento.

As atividades de QA devem ser integradas com o plano de projeto que implementa um ou mais modelos de desenvolvimento de software. O planejamento do QA deve incluir uma lista de atividades necessárias para garantir qualidade no projeto. Além disso, para cada uma das atividades deve ser determinado tempo, tipo, responsável e recursos (Galin, 2003). A intensidade das atividades de QA é afetada por fatores como tamanho do projeto, complexidade técnica, criticidade, qualificação da equipe entre outros.

As atividades de QA devem estar divididas em duas equipes. A equipe de desenvolvimento do produto que deve realizar o trabalho técnico e a equipe de garantia de qualidade que deve ser responsável pelo planejamento de qualidade, por manter ativos de projeto para análise, realizar análises de qualidade e reportar os resultados. Enquanto engenheiros de software tratam qualidade através da aplicação de técnicas e mensurações, conduzindo revisões formais e executando testes bem planejados, o papel da equipe de QA é auxiliar o time de desenvolvimento a alcançar um produto de alta qualidade. A equipe de QA deve:

- Preparar o plano de qualidade - O plano de qualidade deve ser elaborado junto ao plano de projeto e deve ser revisado por todas as partes interessadas. O plano deve identificar as avaliações a serem realizadas, auditorias e revisões, padrões de projeto a serem seguidos, procedimentos para se reportar erros, documentos a serem produzidos como provas das avaliações de qualidade e seus resultados, e forma e data de realização de feedbacks para o time de desenvolvimento.

¹QC - Do inglês Quality Control.

²QA - Do inglês Quality Assurance.

- Participar no projeto de descrição do processo de software - Quando o time de desenvolvimento seleciona, ou lhe é atribuído, um processo de desenvolvimento, a equipe de QA deve revisar o processo, verificar sua compatibilidade com as políticas organizacionais, padrões internos de desenvolvimento, padrões externos de desenvolvimento e outros impactos no planejamento do projeto. A equipe de QA deve também identificar e documentar os desvios do processo e verificar se os ajustes necessários foram realizados.
- Auditar produtos para verificar conformidade com projeto - A equipe de QA deve revisar o produto criado, identificando e documentando desvios e as correções realizadas para os mesmos. Periodicamente devem ser entregues relatórios com os resultados das avaliações aos gerentes.
- Garantir que os desvios sejam documentados e geridos de acordo com o processo utilizado - Todo desvio encontrado no plano de projeto, na descrição do processo, na aplicação de padrões deve ser reportado, e acompanhado até que o problema seja resolvido.
- Controlar as mudanças - A equipe de QA deve coordenar as mudanças realizadas no projeto e realizar coleta e análise e ajuste das métricas de software caso necessário.

Considerando as responsabilidades da equipe de QA, podemos dizer que essas atividades são de extrema importância para a garantia de qualidade dos produtos de software. As subseções 2.4.1, 2.4.2 e 2.4.3 abordam mais detalhadamente algumas atividades consideradas importantes, como revisões, refatoração e testes.

2.4.1 Revisões

A IEEE define revisão como um processo ou reunião na qual um produto ou conjunto de produtos é apresentado à equipe de projeto, gerentes, usuários, consumidores ou partes interessadas para comentários e aprovações. Seus tipos são: revisão de código, de planejamento, formal, de requisitos ou de conclusão dos testes (IEEE, 1991).

De acordo com SWEBOK de Abran e Moore (2004), existem cinco tipos de revisão ou auditorias que são apresentadas no padrão de revisão de software IEEE1028-97 e estão exemplificados na Tabela 2.2.

Tabela 2.2: Tipos de revisão de qualidade

Tipo de revisão	Descrição
Revisão de Gestão	Utilizada para monitorar progresso dos projetos, determinar estado dos planos e cronogramas, confirmar requisitos e sua alocação no sistema, validar a efetividade da abordagem de gestão utilizada para atingir o objetivo.
Revisões Técnicas	Validar um produto de software com base na sua adequação aos propósitos do produto.
Inspeções	Técnica de análise estática visual de produtos de desenvolvimento para detecção de erros, violações de padrão e outros problemas.
Walk-throughs	Técnica de análise estática onde o programador demonstra partes de código ou documentação para os desenvolvedores, ou outras partes interessadas no projeto. Geralmente são menos formais que inspeções.
Auditoria	Fornece uma avaliação independente sobre a conformidade dos produtos e processos, a padrões regulamentários, planos e procedimentos.

2.4.2 Refatoração

Refatorar é o processo de alterar um produto de software para melhorar sua estrutura interna, sem alterar o comportamento externo, podendo ser aplicada a qualquer documento ou parte do produto de *software*. Diferentemente das outras atividades de QA, a refatoração é uma atividade realizada pela equipe de projeto. “Qualquer um pode escrever um código que um computador consegue entender. Bons programadores escrevem códigos que serem humanos possam entender” (Fowler, 2005). A refatoração pode ser realizada separando um método em outros menores e mais coesos, trocando forma de acesso a variáveis e outras alterações que em conjunto ajudam a melhorar a arquitetura, e portanto a qualidade do software.

Fowler (2005) define refatoração como uma alteração realizada na estrutura interna de um software que o torna mais legível e barato de modificar. Não se encontra na literatura estudos que mostrem que refatoração irá solucionar todos os problemas de um produto de software, e nem é esse seu objetivo, mas ainda sim a atividade é de extrema importância e bem vista por muitos gerentes e equipes de projeto (Abbas, 2009). Melhorar o projeto de um software o tornará mais compreensível, auxiliará na detecção e, principalmente, na correção de falhas.

2.4.3 Teste de Software

Uma vez que a atividade de teste de software contribui para o aumento da confiança nos produtos desenvolvidos, o teste de software influencia diretamente a qualidade de um produto. O teste de software por si só não aumenta a qualidade de um sistema, mas serve com um mecanismo para se controlar e aferir se os requisitos do produto estão sendo alcançados. A atividade de teste contribui para evidenciar que um software desempenha as funções de acordo com suas especificações (Pressman, 2010). É impossível afirmar com certeza que um software estará livre de defeitos, porém o teste de software contribui para que haja um aumento da confiança, através de indicadores que demonstram que o software executa as funcionalidades esperadas de forma aceitável e com o mínimo de qualidade (Harrold, 2000).

A atividade de teste, segundo Delamaro et al. (2007); Maldonado (1991), pode ser dividida em três fases, sendo essas: teste de unidade, teste de integração e teste de sistema.

- **Teste de Unidade** - O teste em nível de unidade busca encontrar defeitos de lógica e de implementação, inseridos na menor unidade de software, visando garantir o bom funcionamento do mesmo (Myers et al., 2004).
- **Teste de Integração** - O teste de integração é utilizado na integração de módulos de software e tem por objetivo revelar defeitos relacionados a comunicação e interfaces (Maldonado, 1991).
- **Teste de Sistema** - Após integração dos módulos do sistema, é realizado um teste geral do programa desenvolvido com objetivo de avaliar o sistema como um todo, observando se funcionalidades descritas nas especificações do sistema foram corretamente implementadas (Pressman, 2010).

Além das fases apresentadas, existe ainda uma atividade denominada teste de regressão. O teste de regressão é uma atividade a ser utilizada não durante o desenvolvimento, mas na manutenção do software (Delamaro et al., 2007).

O teste completo de *software* é uma atividade impraticável devido ao seu alto custo computacional e financeiro (Myers et al., 2004). Por este motivo, a busca por outras formas de se selecionar um subconjunto de casos de teste para execução de um programa, que possua alta probabilidade de revelar defeitos, constitui uma tarefa de grande importância (Myers et al., 2004).

Para cada seleção desse subconjunto, são estabelecidas regras denominadas critérios de teste. Por meio dos critérios de teste são selecionados os casos de teste que devem pertencer ao subdomínio (Delamaro et al., 2007). De modo geral são definidos requisitos de teste que, quando satisfeitos, determinam a inclusão de casos de teste em um subdomínio.

Pode-se identificar três técnicas de teste nos quais os critérios se inserem (Delamaro et al., 2007): a funcional, a estrutural e a baseada em defeitos. O que diferencia essas técnicas é o tipo de informação utilizada para determinar os subdomínios. Cada técnica possui características positivas, negativas e um contexto de aplicação.

- **Teste Funcional:**

No teste funcional, também conhecido como teste de caixa-preta, os critérios e requisitos de teste são obtidos através da especificação funcional do sistema (Delamaro et al., 2007). Desta forma são consideradas apenas as entradas, saídas e estado do programa, não sendo necessário ao testador ter conhecimento do código fonte do software (Myers et al., 2004). O principal objetivo da aplicação desta técnica de teste é a encontrar diferenças entre comportamento atual de parte do sistema e sua especificação.

- **Teste Estrutural:**

No teste estrutural, também conhecido como caixa-branca, os casos de teste são derivados a partir da estrutura interna do sistema (Myers et al., 2004; Pressman, 2010). Esses critérios podem ser divididos ainda em critérios baseados em fluxo de controle (Zhu et al., 1997), e em fluxo de dados (Maldonado, 1991; Rapps e Weyuker, 1982). Nos critérios baseados em fluxo de controle são utilizadas informações referentes a execução do programa, como por exemplo, comandos e desvios. Os critérios baseados em fluxo de dados requerem que os caminhos que envolvam definições e usos de variáveis sejam testados. Esses critérios investigam os modos nos quais os valores são associados as variáveis do programa e como essas associações afetam a sua execução (Zhu et al., 1997).

- **Teste Baseado em Defeitos:**

A técnica baseada em defeitos utiliza informações sobre os tipos de erros mais frequentemente cometidos por programadores no processo de desenvolvimento de software. O critério mais conhecido desta técnica é a análise de mutantes (Budd, 1981; DeMillo et al., 1978).

2.5 Métricas de Qualidade

Uma métrica de qualidade é uma propriedade mensurável que indica se um ou mais critérios de qualidade estão sendo alcançados (Gillies, 2002a). Métricas podem auxiliar a prever ou descrever o estado de um produto de software, dependendo de sua fase no ciclo de vida.

Kan (2002) classificou as métricas em três categorias: métricas de produto, métricas de processo e métricas de projeto. Além disso, Kan (2002) afirma que métricas de qualidade são um subconjunto de métricas de software que focam em aspectos como: tempo médio para falha, densidade de defeitos, problemas com consumidor e satisfação do consumidor.

Humphrey (1987) fornece uma lista de quarenta métricas disponíveis na literatura. O problema dessas métricas é que apesar de ser um grande número, elas não estão uniformemente distribuídas entre fatores de qualidade conforme mostra a Tabela 2.3. Podemos perceber que uma grande concentração de métricas de confiabilidade e manutenibilidade, porém não existem métricas para eficiência, adaptabilidade, interoperabilidade e reusabilidade.

Tabela 2.3: Métricas por fator de qualidade (Humphrey, 1987)

Fator de qualidade de software	Quantidade de métricas
Manutenibilidade	18
Confiabilidade	12
Usabilidade	4
Corretude	3
Integridade	1
Escalabilidade	1
Portabilidade	1
Eficiência	0
Adaptabilidade	0
Interoperabilidade	0
Reusabilidade	0

Gillies (2002a) afirma que as métricas disponíveis na literatura são limitadas pois: muitas dessas métricas não podem ser validadas, geralmente não são objetivas, a qualidade é relativa e não absoluta e as métricas não aferem a qualidade em sua totalidade. O modelo Objetivo/Questão/Métrica (GQM³) foi criado na tentativa de relacionar de forma mais direta e simples os fatores de qualidade as métricas disponíveis.

O modelo GQM foi proposto por Basili et al. (1994) e é baseado na premissa de que a empresa precisa definir suas metas e as metas de seus projetos para que a mesma possa medir a qualidade de seus produtos. A organização deve monitorar os objetivos e fornecer um *framework* para analisar os dados coletados (Basili et al., 1994). A ideia geral é refinar os objetivos através de perguntas. Essas perguntas devem ser divididas em questões menores até que ela se torne objetiva e seja possível extrair uma métrica direta que pode ser objetiva ou subjetiva (Abbas, 2009), conforme mostra a Figura 2.5.

Se um dos objetivos do sistema é ser eficiente, devem ser criadas perguntas acerca da eficiência do sistema e como garanti-la. Essas perguntas devem ser formuladas de modo

³do inglês Goal/Question/Metric

Figura 2.5: Estrutura do modelo hierárquico GQM (Basili et al., 1994)

que possam ser respondidas diretamente por métricas que refletem os objetivos do sistema. Uma mesma métrica pode ser utilizada para se responder diferentes questões sobre um mesmo objetivo, e uma questão em grande parte das vezes não pode ser respondida por apenas uma métrica.

Apesar de existirem muitas medidas de qualidade de software, corretude, manutenibilidade, integridade, usabilidade fornecem poderosos indicadores para a equipe de desenvolvimento de um projeto (Pressman, 2010).

- Corretude - Um programa deve operar corretamente. Corretude é o grau no qual um software realiza as funções esperadas. A métrica mais comum de corretude é o número de defeitos por KLOC, onde um defeito é definido por uma falta de conformidade com a especificação. Quando se considera a qualidade do produto como um todo, defeitos são os problemas reportados pelos usuários do programa, após o lançamento do mesmo para o público alvo.
- Manutenibilidade - Manutenção de software requer mais esforço do que qualquer outra atividade de engenharia de software. A manutenibilidade é a facilidade com a qual um programa pode ser corrigido quando um erro é encontrado, quando o ambiente sofre alterações, ou quando o cliente deseja mudar os requisitos do sistema. Não existe uma forma direta de se aferir manutenibilidade, portanto devem ser utilizadas métricas indiretas. Uma métrica simples orientada a tempo é a MTTC⁴ (tempo médio para mudança), que consiste no tempo necessário para se analisar uma requisição de mudança, projetar a modificação, implementá-la, testá-la e distribuí-la aos usuários. Considerando domínios semelhantes e tipos de alterações semelhantes, teremos que, na média, programas que possuem maior manutenibilidade irão possuir menor MTTC do que programas com baixa manutenibilidade.
- Integridade - A integridade de um software é cada vez mais importante (Pressman, 2010). Esse atributo mede a habilidade de um sistema de suportar ameaças (intenci-

⁴do inglês mean-time-to- change

onais ou não) a sua segurança. Ataques podem ser realizados em três componentes: programas, dados e documentos. Para aferir a integridade, dois novos atributos devem ser definidos: ameaça e segurança. Ameaça é a probabilidade de um ataque específico ocorrer (deve ser estimada baseada em evidências empíricas). Segurança é a probabilidade de que um ataque específico será repellido (deve ser estimada de dados empíricos). A integridade de um sistema pode ser definida como

$$Integridade = \sum_{i=0}^n [(1 - P_{ameaca})X(1 - P_{seguranca})];$$

onde n representa o número de tipos de ataque, ou seja ameaça e segurança são iterados sobre cada tipo de ataque.

- Usabilidade - Usabilidade já é uma palavra intrínseca dos produtos de software. Se um produto não é de fácil utilização, ele está predestinado a falhar como produto, mesmo que execute as funcionalidades requeridas. Usabilidade pode ser medida por quatro características:

1. As habilidades físicas e intelectuais necessárias para se entender o sistema.
2. Tempo necessário para se tornar eficiente no sistema.
3. Aumento real na produtividade (comparado com o problema que o sistema resolve) medido quando o sistema é utilizado por alguém já eficiente nele.
4. Elicitação de informações dos usuários sobre o sistema.

- Eficiência na remoção de defeitos

A eficiência na remoção de defeitos (DRE⁵) é uma métrica que merece um destaque pois fornece benefícios qualidade no nível de projeto e de processo. Considerando o projeto como um todo, temos que DRE é:

$$DRE = \frac{E}{(E + D)}$$

onde E é o numero de erros encontrados antes da entrega do software para o usuário final e D é o numero de defeitos encontrados após a entrega. O valor ideal do DRE é 1, o que indica que não foram encontrados defeitos após a entrega do software. Na prática D será maior que 0. Quanto maior o valor de E, é provável que o valor de D seja menor.

⁵Do inglês Defect Removal Efficiency

Se utilizada como métrica que fornece um indicador sobre a habilidade de se encontrar erros nas atividades de QC e QA, DRE irá encorajar a equipe de projeto a implementar técnicas para encontrar o maior número de erros possível.

DRE também pode ser utilizada dentro de projetos para avaliar a habilidade de uma equipe em encontrar erros, antes do projeto passar para uma fase subsequente. O objetivo de qualidade para uma equipe de desenvolvimento de software deve ser atingir um DRE próximo de 1, ou seja, os erros devem ser encontrados antes de passar para a próxima atividade.

2.6 Métodos Ágeis

Muitos estudos, revisões, e pesquisas foram realizadas sobre métodos ágeis, sempre com o foco em validar a melhoria da qualidade e velocidade de execução dos projetos (Abrahamson et al., 2002; Cohen et al., 2004; Larman, 2004; Levine, 2005)

O termo ágil se refere à metodologia de desenvolvimento de software (Fowler, 2005). O termo foi criado durante uma reunião onde dezessete dos defensores de abordagens mais leves de desenvolvimento de software, em comparação com as abordagens tradicionais de desenvolvimento, se reuniram em um *workshop* em 2001 (Cockburn et al., 2001). Antes dessa reunião, diferentes grupos já vinham desenvolvendo independentemente métodos e práticas para reagir frente às mudanças enfrentadas no desenvolvimento de software (Cohen et al., 2004; Fowler, 2005).

São consideradas neste trabalho as seguintes definições de métodos e metodologias. Um método é um programa que regula previamente uma série de operações que se devem realizar, apontando erros evitáveis em vista de um resultado determinado. Metodologia é o estudo de métodos, é a explicação de forma rigorosa e exata de toda ação desenvolvida em um método. Neste sentido, o termo metodologia será usado para a definição do processo ágil, suas práticas e seus conceitos. Método será empregado para o mapeamento da metodologia em um processo específico, por exemplo, XP ou SCRUM.

Os métodos ágeis aplicam desenvolvimento iterativo e evolutivo, planejamento adaptável e flexível, promovem entregas evolutivas e incluem outros valores e práticas para se promover a agilidade, rapidez e respostas flexíveis as mudanças (Larman, 2004). Iterações curtas com refinamento adaptável e evolucionário do planejamento são práticas básicas comuns entre grande parte dos métodos ágeis (Beck e Andres, 2004; Larman, 2004).

No ano de 2001, um grupo interessado em métodos ágeis e iterativos formaram a Aliança Ágil⁶, trazendo um manifesto e uma declaração de princípios (Larman, 2004). O manifesto reuniu representantes de diferentes metodologias ágeis que vinham sendo

⁶www.agilealliance.com

propostas, como *Extreme Programming* (XP), SCRUM, Crystal entre outros, além de pessoas interessadas em novas abordagens diferentes das anteriores, orientadas a extensa documentação e muitas vezes morosas.

O manifesto promove melhores formas de se desenvolver software, através de seus valores (Cockburn et al., 2001) :

- Indivíduos e interações mais que processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

Os valores podem ser traduzidos nos doze princípios da metodologia ágil também listados por Cockburn et al. (2001):

1. Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2. Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3. Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4. Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5. Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6. O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7. Software funcionando é a medida primária de progresso.
8. Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9. A contínua atenção à excelência técnica e bom projeto aumenta a agilidade.

10. Simplicidade - a arte de maximizar a quantidade de trabalho não realizado - é essencial.
11. As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizáveis.
12. Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

É importante reforçar a ideia de que o movimento ágil não é uma anti-metodologia. É apoiada a utilização de modelagens, mas não a elaboração de diagramas que nunca serão utilizados. A documentação é importante, entretanto não se deve criar centenas de páginas que raramente serão lidas e mantidas. É necessário que haja um planejamento, porém que considere um ambiente suscetível a mudanças (Boehm e Turner, 2003; Larman, 2004).

Outro fator importante a ser considerando é que as ideias utilizadas pelos métodos ágeis não são exatamente inovadoras. Elas surgiram da reutilização de ideias, advinda da experiência das pessoas como uma reação às abordagens tradicionais de desenvolvimento de software.

- **Reação as abordagens tradicionais**

Com a crescente mudança nos modelos de negócio e no ambiente corporativo, os métodos de desenvolvimento precisam se adaptar mais rapidamente a essas mudanças. De acordo com Cockburn e Highsmith (2001b), os métodos ágeis foram propostos de uma perspectiva que espelha as mudanças turbulentas tecnológicas e de negócio de hoje em dia. As abordagens tradicionais não lidavam com essas mudanças, assumindo que seria possível implementar um conjunto significativos de requisitos no ciclo de vida de um software antes de se fazer alguma alteração.

- **Reutilização de ideias**

Muitas das ideias utilizadas nos métodos ágeis não são novas (Abbas, 2009). Além disso, muitas pessoas acreditavam que essa era a forma mais bem sucedida de se desenvolver software. Entretanto, essas ideias nunca foram levadas a sério, e nunca tinham sido apresentadas como um todo, como uma metodologia completa (Larman, 2004).

Ideias de métodos ágeis existiam em outros processos de desenvolvimento que pregavam entregas rápidas, iterações curtas e abordagens diferentes para aderir a mudanças. Empresas como a DuPont apresentaram métodos como o RIPP (do inglês *Rapid Iterative Production Prototyping*). James Martin expandiu este método em

um maior e mais formalizado que se tornou o RAD (do inglês *Rapid Application Development*) (Martin, 1991). O RAD possui muitas ideias que são utilizadas nos métodos ágeis. RAD recomenda entregas rápidas, desenvolvimento iterativos, times pequenos e altamente treinados e envolvimento do cliente em cada estágio.

- **Experiências pessoais**

As pessoas envolvidas no manifesto ágil vivenciaram situações diferentes no desenvolvimento de software. O manifesto então reuniu pessoas que propunham outras alternativas baseados em suas próprias experiências (Fowler, 2005). Kent Beck, o fundador do XP havia sido contratado para contribuir em um projeto de sistema de folha de pagamento. Ken Schwaber, um dos fundadores do SCRUM era dono de uma empresa e tinha dificuldades em lidar com requisitos que mudavam o tempo todo. As experiência desses profissionais fez com que buscassem maneiras alternativas de se controlar o processo de desenvolvimento de software (Abbas, 2009).

A seguir são descritos os principais métodos ágeis existentes.

2.7 Métodos Ágeis Utilizados na Pesquisa

Sob o termo guarda-chuva ágil, existem diversas abordagens que implementam suas ideias e princípios. Entre os métodos ágeis mais conhecidos pode-se citar o *Extreme Programming* (XP) (Beck e Andres, 2004), SCRUM (Schwaber e Beedle, 2001), *Adaptive Software Development* (ASD) (Highsmith, 2000), *Crystal Methods* (Cockburn, 2005) entre outros.

No início deste projeto foi realizado um pequeno levantamento informal sobre quais métodos ágeis são mais utilizados no mercado brasileiro de desenvolvimento de software. Foram identificados inicialmente o SCRUM e o XP. Com o desenvolvimento da pesquisa chegou-se a conclusão de que esses dois métodos deveriam ter um enfoque maior, dado que muitas empresas nacionais trabalham com os mesmos. Além disso, o enfoque nesses dois métodos facilitou os trabalhos empíricos desta pesquisa, uma vez que as empresas participantes do projeto utilizavam esses métodos, ou adaptações dos mesmos.

2.7.1 SCRUM

O SCRUM foi criado quando Ken Schwaber e Jeff Sutherland perceberam que o desenvolvimento de software é uma atividade imprevisível. Assim como o XP, o SCRUM é um dos métodos ágeis mais utilizados (Ambler, 2005). Este método ágil define um *framework* de

gestão, conduzido pelo *SCRUM Master*⁷. SCRUM é o único método ágil que foi escalado para projetos de tamanho médio (Boehm e Turner, 2003).

No SCRUM, a iteração dura 30 dias e é chamada de *sprint*. O *sprint* será precedido por um planejamento pré-*sprint* e será seguido por uma reunião pós-*sprint*. Toda a equipe deve trabalhar nos requisitos definidos no início de cada *sprint* (Abrahamsson et al., 2002; Schwaber e Beedle, 2001).

2.7.1.1 Práticas

O SCRUM não define práticas específicas para o processo de desenvolvimento de software focando apenas no processo de gestão do desenvolvimento. As práticas de gestão do SCRUM são (Abrahamsson et al., 2002):

- Lista de *Backlog* de Produto - Lista de requisitos técnicos e de negócios, ordenados por prioridade. Pode incluir funcionalidades, correções, atualização de tecnologia entre outras atividades. Esta lista deve ser controlada através da inserção, remoção e atualização de itens durante o processo de desenvolvimento.
- Estimativa de esforço - Análise detalhada dos itens do *backlog*, com estimativa de esforço necessário para se cumprir a atividade.
- *Sprint* - São as iterações realizadas durante o desenvolvimento. Cada iteração corresponde a um ciclo composto por variáveis de ambiente as quais são modificadas a cada iteração. A equipe deve se organizar para produzir um incremento de software executável e a iteração deve durar aproximadamente trinta dias.
- Reunião de planejamento de *sprint* - É realizada antes do início de um novo *sprint*. Ela é composta por duas fases, onde na primeira fase o cliente seleciona itens do *backlog* e os prioriza. Em um segundo momento a equipe do projeto planeja como o incremento será produzido.
- *Backlog* de *sprint* - Lista de itens selecionados do *backlog* a serem desenvolvidos em um determinado *sprint*.
- Reunião diária - Reunião para verificar o progresso diário do projeto. A pauta da reunião deve ser sempre as atividades realizadas no dia e quais atividades serão realizadas até a próxima reunião. A duração da reunião deve ser rígida e ter no máximo 15 minutos.

⁷*SCRUM Master* - Pessoa responsável por interagir equipe de desenvolvimento e cliente, mantendo a visão geral do projeto

- Reunião de revisão de *sprint* - Realizada no final de um *sprint*, onde a equipe apresenta os resultados finais da iteração realizada. Esses resultados podem gerar mais itens no *backlog* do produto e esses devem ser considerados no planejamento do próximo *sprint*.

2.7.1.2 Processo

O ciclo de vida de um projeto que segue o método SCRUM possui três fases: pré-jogo, desenvolvimento e pós-jogo (Abrahamsson et al., 2002). A figura 2.6 detalha as três fases do processo de desenvolvimento SCRUM junto com as atividades inerentes a cada fase (Abrahamsson et al., 2002).

Figura 2.6: Processo de Desenvolvimento SCRUM (Abrahamsson et al., 2002)

Fase de pré-jogo

A fase de pré-jogo (ou em inglês, *pregame phase*) é dividida em duas subfases: planejamento e desenho da arquitetura em alto nível.

O planejamento consiste em definir o produto a ser desenvolvido. Uma lista de *backlog* é criada contendo todos os requisitos conhecidos até o momento. Os requisitos podem ser originados de clientes, divisão de marketing e vendas e até desenvolvedores de software. A lista de *backlog* é constantemente atualizada com informações mais novas, mais detalhadas, com estimativas mais acuradas e prioridade definida. A fase de planejamento também inclui definição de equipe, ferramentas, recursos, elicitação de riscos, necessidade de treinamento e aprovações de gestão. A cada *sprint* a lista de *backlog* é revista pela equipe.

Na subfase de desenho da arquitetura, a arquitetura é planejada em alto nível considerando os itens iniciais existentes no *backlog*. No caso de melhoria em um projeto existente

essa subfase tem como objetivo identificar os problemas que o desenvolvimento das histórias do *backlog* podem causar. É realizada então uma reunião de revisão com o propósito de definir como proceder com as alterações, caso necessárias.

Fase de desenvolvimento

A fase de desenvolvimento (ou em inglês, *game phase*) é a parte ágil da abordagem SCRUM (Abrahamsson et al., 2002). Nesta fase o sistema é desenvolvido em *sprints*. Os *sprints* são ciclos iterativos onde as funcionalidades são desenvolvidas ou melhoradas para produzir novos incrementos (Abrahamsson et al., 2002; Schwaber e Beedle, 2001). Cada *sprint* inclui as fases tradicionais de desenvolvimento de software: requisitos, análise, projeto, desenvolvimento, evolução e entrega. A arquitetura e o planejamento do sistema evoluem durante o desenvolvimento de um *sprint*, que deve durar de uma a quatro semanas.

Variáveis técnicas e do ambiente, como tempo, qualidade, requisitos, recursos e ferramentas são observadas durante a fase de desenvolvimento. Ao invés de se preocupar prematuramente com esses problemas, o SCRUM foca no controle dessas variáveis durante o processo de desenvolvimento, tornando assim o processo mais flexível a mudanças (Abrahamsson et al., 2002).

Fase de pós planejamento

A fase de pós planejamento (em inglês, *postgame phase*) contém as atividades para encerramento de uma versão. Ela se inicia quando é acordado entre as partes (cliente e equipe de desenvolvimento) que as variáveis de ambiente, como requisitos estão completas. Neste caso, não restam histórias a serem desenvolvidas e não será criado mais nenhuma nova história. O sistema, ou versão, está pronto para lançamento e essa preparação é feita na fase de pós planejamento, incluindo tarefas como integração, teste e documentação.

2.7.1.3 Papéis e responsabilidades

Existem seis papéis identificados no SCRUM que possuem diferentes tarefas e propósitos durante o processo e suas práticas: o SCRUM Master, Proprietário, Equipe SCRUM, Cliente, Usuário e Administrador. A seguir, esses papéis são apresentados de acordo com as definições de (Schwaber e Beedle, 2001):

- SCRUM master - É um novo papel administrativo introduzido pelo SCRUM. Ele é responsável por garantir que as práticas, valores e regras do SCRUM sejam adotadas e executadas durante todo o projeto, sendo também encarregado por remover ou alterar qualquer impedimento ao processo de desenvolvimento, permitindo que a

equipe trabalhe da forma mais produtiva possível. Ele interage com a equipe, com o cliente e com o administrador, gerenciando essa comunicação durante todo o projeto.

- **Product Owner** - Ele é escolhido pelo SCRUM Master, pelo cliente e pelo gerente, sendo oficialmente responsável pelo gerenciamento, controle e visibilidade do *backlog*. Ele é responsável pelas decisões finais sobre o *backlog*, participando na estimativa de tempo e esforço para os itens de *backlog* e transformando-os em funcionalidades a serem desenvolvidas.
- **Equipe SCRUM** - A equipe de desenvolvimento SCRUM possui a autoridade para decidir sobre ações necessárias, se organizando para atingir as metas de cada *sprint*. Dessa forma, a equipe está envolvida em atividades como por exemplo, estimativa de esforços, criação e revisão do *backlog* e sugestão de impedimentos que precisam ser removidos para o bom desenrolar do projeto.
- **Cliente** - Os cliente participam das tarefas relacionadas aos itens de *backlog* de produto, para que o sistema seja desenvolvido ou melhorado.
- **Gerente** - O gerente é encarregado pelas decisões finais sobre contratos, padrões e convenções a serem seguidas no projeto. Ele também participa das definições dos objetivos e dos requisitos.

2.7.2 Extreme Programming

O método XP⁸ é originário de problemas decorrentes de processos de desenvolvimento que utilizavam abordagens tradicionais de desenvolvimento de software.(Beck, 1999a). Reunindo algumas práticas dos processos de desenvolvimento de software e após alguns testes bem sucedidos foi formulada a teoria do método XP, com seus princípios chave e práticas utilizadas. Suas práticas não eram novas, entretanto a escolha dessas práticas e sua organização formaram um novo método de desenvolvimento de software (Abrahamsson et al., 2002; Beck, 1999b; Larman, 2004).

Os quatro valores do método ágil XP são (Teles, 2005):

1. *Feedback*

A elicitação de requisitos é uma das atividades mais complicadas do processo de desenvolvimento de software. Ela guia os esforços do projeto e por isso o entendimento dos requisitos e necessidades dos clientes passam a ter extrema importância no processo de desenvolvimento (Teles, 2005).

⁸do inglês *Extreme Programming*

Brooks (1987) afirma que não é possível para os clientes prever corretamente as funcionalidades necessárias para o software que os atenderá. Por esse motivo é necessária uma grande interação entre desenvolvedores e cliente ao longo do projeto. A compreensão das necessidades dos usuários é um aprendizado contínuo no qual os desenvolvedores aprendem sobre os problemas do negócio e os usuários tomam conhecimento das dificuldades de limitações técnicas (Teles, 2005).

Por este motivo o XP é organizado em pequenos ciclos de desenvolvimento, para que haja um *feedback* constante. O objetivo passa ser apresentar a funcionalidade ao usuário rapidamente, para que ele possa detectar falhas e desvios o mais cedo possível. Desta forma o XP permite que erros das pessoas sejam descobertos relativamente cedo e reparados de forma metódica e a um menor custo (Cockburn, 2002).

2. Comunicação

Os projetos de software, em sua maioria, envolvem pelo menos duas pessoas no processo de desenvolvimento, o usuário e o desenvolvedor. De fato, projetos de software envolvem grupos de pessoas geralmente não homogêneo e com processo de comunicação não bem definido ou utilizado. A transmissão de conhecimento tácito representa um desafio para as equipes de desenvolvimento (Abrahamsson et al., 2002; Teles, 2005). Para solucionar esses problemas, os processos de desenvolvimento propõe mecanismos de comunicação mais ou menos eficazes a serem adotados nos projetos.

Para sanar esse tipo de problema, o método ágil XP procura envolver ativamente os usuários, tornando-os membros da equipe de desenvolvimento e fazendo-os compartilhar a mesma sala física de projetos. Desta forma a comunicação é facilitada, pois é constante e direta (Cockburn, 2002).

3. Simplicidade

É comum observar em projetos de desenvolvimento de software que a arquitetura ou funcionalidades criadas para o sistema já é preparada para problemas futuros que ainda não existem. Como o escopo é definido no início do projeto, os desenvolvedores criam soluções genéricas evitando ou tentando minimizar o impacto das alterações que possam ocorrer no escopo. Além disso é comum desenvolvedores adicionarem funcionalidades excedentes com intuito de antecipar o futuro, tentando prever necessidades do cliente.

No XP os desenvolvedores procuram implementar funcionalidades de acordo com suas prioridades para cada iteração com foco apenas no que é essencial. Generalizações que não estejam explicitamente descritas como necessárias não são implementadas, tentando manter a simplicidade das funcionalidades (Jeffries et al., 2001).

4. Coragem

Observa-se nos processos de desenvolvimento de software alguns temores dos clientes que podem atrapalhar desenvolvimento. O cliente teme não obter o que solicitou, ou fazer uma solicitação que leva a uma implementação incorreta, gastará mais recursos do que o necessário ou planejado (Beck e Fowler, 2001).

As equipes XP devem reconhecer e enfrentar esses medos. Para isso deve-se confiar nos mecanismos que o método proporciona para alcançar o sucesso do projeto. Um exemplo de mecanismo que auxilia na mitigação desses riscos são as iterações curtas. Mesmo que o cliente descreva o projeto de forma errada, ou a equipe de programação não implemente as funcionalidades esperadas, o tempo para *feedback* é curto, fazendo com que problemas sejam descobertos com rapidez e com que os problemas não se propaguem (Teles, 2005).

2.7.2.1 Práticas

O XP foca em possibilitar o desenvolvimento de software com sucesso, apesar de requisitos vagos ou em constante alteração. Iterações curtas com pequenos lançamentos e *feedback* rápido, participação do cliente, comunicação e coordenação, integração contínua e testes, documentação limitada e programação pareada, estão entre as principais características do XP (Abrahamsson et al., 2002). Em Beck e Andres (2004) apresentam as práticas do XP:

- Planejamento - A equipe do projeto estima o tempo de implementação das histórias, definindo a data de entrega da iteração.
- Entregas frequentes - As entregas devem ser frequentes para garantir um rápido *feedback* das funcionalidades desenvolvidas. Além disso a entrega constante é um fator positivo motivacional para a equipe de desenvolvimento.
- Metáfora - O sistema deve ser definido através de uma, ou um conjunto de metáforas entre cliente e equipe de desenvolvimento. Essas histórias guiarão o desenvolvimento, descrevendo como o sistema funciona.

- Simplicidade - Ênfase no desenvolvimento da solução mais simples a ser implementada no momento. Complexidades desnecessárias e códigos extras devem ser removidos.
- Desenvolvimento orientado a testes - Os testes devem ser escritos antes do código, e devem ser executados continuamente. Os clientes devem escrever os testes funcionais.
- Refatoração - Melhorar o código, através da remoção de código duplicado, melhoria de comunicação, simplificação de métodos, sem alterar a funcionalidade do sistema.
- Programação pareada - O desenvolvimento do código é feito por dois programadores, permitindo assim uma revisão maior e constante do código gerado.
- Código de propriedade coletiva - Qualquer programador pode trocar qualquer pedaço do código a qualquer momento.
- Integração contínua - Cada funcionalidade implementada é integrada ao sistema. A cada integração os testes devem ser executados novamente para assegurar que a nova funcionalidade não tenha “quebrado” o código já existente.
- Jornada de trabalho - A jornada de trabalho deve ser de no máximo 40 horas semanais. A existência de trabalho fora do horário indica que existe um problema no processo de desenvolvimento ou no planejamento.
- Presença do cliente - O cliente deve estar presente no desenvolvimento do projeto, compartilhando a mesma sala que a equipe de projeto. Desta forma, o *feedback* do cliente é constante assegurando prioridade de requisitos a serem implementados e sanando dúvidas que eventualmente podem surgir sobre o sistema.
- Código padronizado - Deve ser definido um padrão de regras de codificação que devem ser seguidas à risca. A comunicação através do código deve ser enfatizada.
- Ambiente de trabalho - O ambiente de trabalho deve favorecer a comunicação da equipe. É recomendado uma sala grande com pequenos cubículos, e as equipes de programação pareadas devem se sentar ao meio.
- Regras - As equipes podem e devem definir suas próprias regras, que podem ser mudadas a qualquer momento. Entretanto as alterações devem ser acordadas entre as equipes e o impacto da mudança deve ser avaliado.

2.7.2.2 Processo

O ciclo de vida do XP consiste em seis fases: Exploração, Planejamento, Iterações, Produção, Manutenção e Morte do projeto. A Figura 2.7 mostra essas fases com suas atividades e práticas, conforme (Beck, 1999b), as quais são descritas a seguir.

Figura 2.7: Ciclo de Vida do Processo XP (Abrahamsson et al., 2002)

Exploração

Nesta fase, os clientes escrevem as histórias que eles gostariam que fossem incluídas na primeira versão a ser lançada. Cada história deve descrever uma funcionalidade a ser adicionada no programa. Paralelamente a equipe de projeto deve se familiarizar com ferramentas, tecnologias e práticas que serão utilizadas no projeto. A tecnologia a ser utilizada deve ser testada e as possibilidades de arquitetura para o sistema exploradas (inclusive com construção de protótipos, caso necessário). A fase de exploração deve levar de poucas semanas a poucos meses dependendo da familiaridade dos programadores com a tecnologia.

Planejamento

A fase de planejamento prioriza as histórias e delimita o escopo da primeira versão do programa a ser liberada. Os programadores devem estimar o tempo necessário para implementação de cada história e assim é possível agendar a data da primeira entrega. O espaço de tempo para a primeira entrega não deve exceder dois meses. A fase de planejamento em si demora apenas alguns dias para ser concluída.

Iterações

Esta fase inclui várias iterações do sistema até que a primeira entrega seja realizada. O cronograma criado na fase de planejamento é quebrado em atividades menores, iterações, que levam de uma a quatro semanas para serem concluídas. A primeira iteração deve focar na arquitetura do sistema. Para isso, devem ser escolhidas as histórias que obriguem os desenvolvedores a implementar a arquitetura básica do sistema. O cliente pode decidir quais histórias serão selecionadas para cada iteração. Os testes funcionais devem ser escritos pelos clientes e são executados ao final de cada iteração. Ao final da última iteração o sistema está pronto para a produção.

Produção

A fase de produção requer testes extras e verificações de performances antes que o sistema seja lançado para uso do cliente. Nessa fase podem ainda surgir algumas alterações que podem ser incluídas no lançamento da próxima versão. Neste caso podem surgir iterações mais curtas, para agilizar a entrega dessas correções. As alterações não aceitas podem ser postergadas, e devem ser documentadas para implementação futura.

Manutenção

Após lançamento da primeira versão, o projeto deve manter ambos, produção e desenvolvimento das próximas iterações. É comum nesta fase haver um aumento da equipe, ou redução da força de entrega nas iterações.

Morte do Projeto

A fase de morte do projeto é quando o cliente não possui mais histórias para serem implementadas. Para isso é necessário que o sistema satisfaça as necessidades do cliente também em outros aspectos como desempenho e confiabilidade. Este é o ponto onde a documentação do sistema deve ser escrita, uma vez que não haverá mais alterações na arquitetura, projeto ou código. Esta fase pode ser antecipada caso o sistema não esteja entregando o resultado esperado ou o orçamento não seja suficiente para continuidade do projeto.

2.7.2.3 Papéis e responsabilidades

Existem diferentes papéis no XP para as diversas tarefas e objetivos durante o processo e suas práticas (Beck, 1999b):

- Programador - Os programadores escrevem os testes e mantém o código do programa o mais simples possível. O primeiro quesito para se obter sucesso no XP é a comunicação e coordenação com outros programadores e membros da equipe.

- Cliente - O cliente determina as histórias e escreve os testes funcionais. Além disso, ele decide quando cada requisito está satisfeito. O cliente define também as prioridades de implementação dos requisitos.
- Analista de Teste - O analista de teste auxilia o cliente na elaboração dos casos de testes funcionais. Regularmente, ele realiza testes funcionais, divulga os resultados e mantém as ferramentas de teste.
- Redator técnico - É o responsável pelo *feedback* no XP. Ele rastreia as estimativas feitas pela equipe e dá *feedback* sobre quão acuradas foram as estimativas, para que assim a equipe se prepare melhor para projetos futuros. Ele também acompanha o progresso de cada iteração e avalia se o objetivo será alcançado dentro dos limites de recursos e tempo, ou se é necessária uma alteração no processo.
- Treinador - Pessoa responsável pelas questões técnicas do projeto, recomenda-se que esta pessoa seja aquela com maior conhecimento do processo de desenvolvimento, dos valores e práticas do XP. É de responsabilidade do treinador verificar o comportamento da equipe frente o processo XP, sinalizando os eventuais erros cometidos pela equipe.
- Consultor - O consultor é um membro externo que detém algum conhecimento técnico específico necessário para o projeto. O consultor guia a equipe para solucionar seus problemas.
- Gerente - Pessoa responsável pelos assuntos administrativos do projeto, mantendo um forte relacionamento com o cliente para que o mesmo participe das atividades do desenvolvimento. O gerente de projeto é responsável por filtrar assuntos não relevantes aos desenvolvedores e aspectos que só serão implementados em iterações futuras. Para um bom exercício de gerente de projeto é necessário que a pessoa conheça e acredite nos valores e práticas do XP para que possa cobrar isto da sua equipe.

Estudo de Caso - Qualidade de Software

3.1 Considerações Iniciais

Com intuito de se entender como empresas ágeis interagem na prática com temas como controle de qualidade e teste de software foi definido um estudo de caso que abrange entrevista, análise de documentos e observação direta. Um estudo de caso é uma investigação empírica de um fenômeno contemporâneo dentro do seu contexto (Yin, 1994).

Para validar o modelo do estudo de caso, um piloto foi realizado. O piloto desse estudo evidenciou algumas dificuldades na obtenção de informação e acesso a documentos, entretanto a principal contribuição do piloto foi a percepção que os pontos a serem observados independem do processo de desenvolvimento.

Desta forma, o estudo foi estendido para uma análise do monitoramento de métricas e atividades de garantia de qualidade para empresas de desenvolvimento de software, independentemente do processo de desenvolvimento utilizado. O protocolo foi alterado para acomodar esse novo modelo, e o estudo foi conduzido com intuito de encontrar diferenças e semelhanças nos diferentes processos de desenvolvimento.

O protocolo e resultados desse estudo de caso são o tema deste capítulo.

3.2 Objetivo

Com o objetivo de investigar a relação entre qualidade do produto e processo de desenvolvimento foi criado um protocolo para um estudo de caso exploratório que rastreia informações sobre métricas e atividades de garantia de qualidade no processo de desenvolvimento, e o impacto das mesmas, na visão da equipe de projeto e na qualidade final do produto.

3.3 Protocolo

O protocolo de um estudo de caso é um arcabouço para todas as definições, decisões e procedimentos a serem utilizados durante o processo de coleta de informações e análises futuras do estudo de caso (Runeson e Höst, 2008). Essa seção descreve o protocolo criado para analisar um estudo de caso sobre qualidade de software e sua relação com o processo de desenvolvimento.

3.3.1 Estratégia de Seleção

Para determinar quais empresas serão objetos deste estudo de caso foram construídas premissas baseadas no processo de desenvolvimento utilizado pelas mesmas.

As empresas necessitam trabalhar com o método de desenvolvimento em questão por pelo menos seis meses. Devem ser selecionadas pelo menos quatro empresas onde metade deve utilizar algum método ágil de desenvolvimento e a outra metade métodos mais tradicionais de desenvolvimento.

Para garantir o que o número mínimo de empresas fosse atingido foi realizado um contato prévio com empresas de desenvolvimento de software inquerindo sobre possibilidade de colaboração com o estudo. Foi contatado também um parque tecnológico que possui um grande número de empresas que possuem foco em métodos ágeis. A seleção das empresas foi feita por conveniência.

Caso a amostra de informações se mostre irrelevante, ou seja, os dados coletados não sejam suficientes para se construir pelo menos uma conclusão, outras empresas poderão ser convidadas para participar do estudo.

Para proteger os entrevistados e empresas participantes nomes e informações sensíveis foram ocultadas dos resultados. Foi assinado um acordo de confidencialidade entre pesquisadores, profissionais envolvidos e empresa.

3.3.2 Triangulação Metodológica

Triangulação é o processo de se abordar o objeto de estudo por diferentes ângulos. Essa técnica é adequada quando se está realizando um estudo qualitativo, pois o material coletado em geral pode ser rico e extenso, porém menos preciso do que dados quantitativos (Runeson e Höst, 2008).

De acordo com Stake (1995), existem quatro tipos de triangulação que podem ser realizadas em um estudo de caso: triangulação de dados, triangulação de observadores, triangulação metodológica e triangulação teórica (Stake, 1995).

Neste trabalho foram utilizados dois tipos de triangulação:

- **Triangulação metodológica:** com o objetivo de validar e de aumentar a confiabilidade dos dados coletados, foram combinadas duas estratégias qualitativas, entrevistas com uma análise de documentos ou observação direta, quando disponibilizado ou autorizados pela empresa participante. Embora somente o uso dessas estratégias não sejam suficiente para conclusões estatisticamente válidas, elas colaboram para a validade dos dados coletados, seja sustentando os dados das entrevistas ou expondo possíveis divergências.
- **Triangulação de observador:** durante a fase de análise dos dados, todo o material coletado durante as entrevistas foi revisado pelo autor com o auxílio de um segundo pesquisador, evitando que as análises fossem influenciadas pelas opiniões pessoais dos pesquisadores.

A utilização da triangulação metodológica no estudo de caso ajudou a mitigar riscos de se obter informações distorcidas ou mal interpretadas.

3.3.3 Entrevistas

Para reunir informações suficientes sobre métricas e garantia de qualidade foi construída uma entrevista semiestruturada para ser conduzida em cada empresa selecionada. Cada empresa indicou dois profissionais para serem entrevistados, um com perfil de gestor ou desenvolvedor com grande experiência e outro com perfil de desenvolvedor.

As entrevistas seguiram um conjunto pequeno de questões para garantir um conjunto mínimo de informações estruturadas. Além do conjunto de questões básicas alguns itens foram selecionados para guiar o rumo da entrevista, uma vez que não é possível prever todos os cenários existentes nas empresas. Considerando este formato, a entrevista semiestruturada auxiliou a explorar a percepção que as pessoas tem sobre a qualidade dos produtos e projetos da empresa onde trabalham.

Um gravador de voz foi utilizado nas entrevistas, com o consentimento prévio dos entrevistados, que tiveram duração de 30 a 40 minutos, garantindo assim que todas as informações das entrevistas pudessem ser analisadas com a profundidade necessária.

As entrevistas foram divididas em três fases:

1. Na primeira fase foi apresentado o propósito do estudo de caso, explicando como os dados seriam utilizados posteriormente. Foi apresentado também uma visão de qualidade de produto de software, seguindo a definição da (IEEE, 1991), para nivelar os conceitos relacionados. Ao final desta fase foi assinado o termo de confidencialidade entre as partes envolvidas.
2. A segunda fase foi conduzida individualmente com cada profissional. O entrevistador na ocasião solicitou que fosse descrito o processo de desenvolvimento da empresa. Em seguida foram aplicadas as questões introdutórias sobre métricas, teste de software e gestão do desenvolvimento.
3. Na ultima fase o entrevistador realizou perguntas mais abertas sobre atividades de controle e garantia de qualidade, métricas específicas de qualidade e ferramentas para avaliação e análise de qualidade.

3.3.4 Ameaças à Validade

A validade de um experimento está relacionada ao nível de confiança que se pode ter no processo de investigação experimental, ou seja, o quão confiáveis são os elementos envolvidos nesse processo, desde a teoria até os resultados obtidos, incluindo apresentação dos resultados e validade de construção interna e externa (Travassos et al., 2002; Wainer, 2007).

Com intuito de garantir maior qualidade ao estudo de caso, foram identificadas possíveis ameaças à validade do estudo e atribuídas a cada uma delas pelo menos uma ação de tratamento. As Tabelas 3.1, 3.2 e 3.3 identificam as ameaças à validade do estudo com possíveis ações de mitigação (marcadas com a letra M), para reduzir o impacto da ameaça ao estudo, ou de contorno (marcadas com a letra C), para reduzir a probabilidade de que a ameaça venha a ocorrer.

A validade de construção considera os relacionamentos entre a teoria e a observação, ou seja, se o tratamento reflete a causa bem e o resultado reflete o efeito bem (Travassos et al., 2002). A Tabela 3.1 descreve as possíveis ameaças à construção que foram identificadas no estudo.

A validade externa de um experimento define condições que limitam a habilidade de generalizar os resultados para a prática industrial (Travassos et al., 2002). A Tabela 3.2 descreve as possíveis ameaças externas que foram identificadas no estudo.

Tabela 3.1: Validade de Construção

Ameaça	Mitigação (M) / Contorno (C)
Interpretação falha por parte do entrevistador.	<ul style="list-style-type: none"> • (M) As entrevistas foram gravadas para que nenhuma informação fosse perdida ou mal interpretada. • (M) A entrevista foi transcrita e o entrevistado recebeu uma cópia para fazer considerações caso necessário.
As informações coletadas não são confiáveis	<ul style="list-style-type: none"> • (M) Quando disponível, documentos foram analisados para corroborar com as informações fornecidas. • (M) Quando possível, foram realizadas observações diretas para corroborar com as informações fornecidas. • (C) Caso houvesse alguma dúvida sobre a qualidade das informações, seja por controvérsia ou falta de evidências, os dados referentes à entrevista foram descartados e não foram considerados para o estudo.
As questões não abordam os temas garantia de qualidade e métricas de qualidade de uma forma aplicável para as empresas participantes do estudo.	<ul style="list-style-type: none"> • (M) Foi realizada uma entrevista piloto para ajustar o protocolo do estudo de caso e as questões a serem realizadas. • (M) As entrevistas foram semiestruturadas, permitindo ao entrevistador adicionar ou descartar questões durante as entrevistas, garantindo a flexibilidade necessárias para casos especiais.

A validade de conclusão está relacionada à habilidade de chegar a uma conclusão correta a respeito dos relacionamentos entre o tratamento e o resultado do experimento, e isso envolve a correta análise e interpretação dos resultados e confiabilidade das medidas (Travassos et al., 2002).

Tabela 3.2: Validade Externa

Threat	Mitigação (M) / Contorno (C)
O conjunto de informações coletadas não é representativo.	<ul style="list-style-type: none"> • (M) Foram entrevistadas mais de um empresa ágil e não ágil para tentar generalizar as conclusões. • (C) Se as informações coletadas não forem suficientes para concluir sobre qualquer tópico da entrevista, outras empresas serão convidadas a participar do estudo de caso. Caso as informações coletadas ainda não forem suficientes, será definido e aplicado um <i>survey</i> estruturado com base na experiência obtida nas entrevistas.

Tabela 3.3: Validade de Conclusão

Threat	Mitigação (M) / Contorno (C)
Respostas obtidas na entrevista são superficiais e não permitem conclusões sobre o processo de desenvolvimento ou atividades de garantia de qualidade e teste de software.	<ul style="list-style-type: none"> • (M) Foram realizadas, sempre que permitido, observações diretas para coletar dados complementares as respostas da entrevista. • (M) Foram realizadas, sempre que permitido, análises de documentações geradas pelo processo de desenvolvimento para coletar dados complementares as respostas da entrevista.

3.3.5 Piloto

Para garantir que as questões elaboradas para a entrevista estão em conformidade com a realidade das empresas entrevistadas no tema garantia de qualidade, foi realizada uma entrevista piloto com uma empresa de desenvolvimento que utiliza o método ágil SCRUM.

A aplicação da entrevista levou a uma adaptação nas questões elaboradas. Duas questões que abordavam métricas de manutenibilidade e confiabilidade foram transformadas em uma pergunta mais ampla pois foi indicado que esses tópicos podem ser avaliados considerando outros fatores além de métricas de manutenibilidade, testabilidade ou quantidade de defeitos.

Além da entrevista piloto, o questionário foi revisado por um especialista na área de qualidade de uma grande empresa de telecomunicações brasileira. A empresa possui mais de seis mil funcionários e utiliza modelo cascata de desenvolvimento, possui uma área dedicada a garantia e controle de qualidade de software. A diretoria da empresa não

permitiu a participação direta no estudo, e a revisão do documento apontou apenas dicas sobre como abordar alguns assuntos.

3.3.6 Questões

As entrevistas seguiram um *checklist* de questões sobre o modelo de desenvolvimento e qualidade de software para reunir um conjunto mínimo necessário de informações. Além disso foram feitas algumas perguntas sobre a empresa a fim de obter dados organizacionais para o estudo. As questões desenvolvidas encontram-se no Apêndice A.

3.4 Coleta de Dados

A coleta de dados foi realizada entre Setembro de 2012 e Março de 2013. Neste período foram contatadas onze empresas para o estudo de caso, entretanto sete foram descartadas. Das empresas descartadas para o estudo, três empresas não permitiram a publicação das informações após a entrevista, duas não puderam fornecer evidências para as informações apresentadas. As outras duas empresas descartadas afirmaram utilizar métodos ágeis para o desenvolvimento de software porém tanto a entrevista quanto a observação direta mostrou que existia uma disparidade muito grande entre os métodos da literatura e o método efetivamente empregado pela empresa.

A tabela 3.4 apresenta alguns dados sobre as empresas participantes para identificar o contexto organizacional em que cada empresa esta inserida. Na sequência é apresentado um resumo do que foi observado e ouvido nas entrevistas com cada empresa.

Tabela 3.4: Empresas selecionadas para o estudo de caso sobre qualidade de software.

Empresa	Método de desenvolvimento	Tamanho da empresa	Produto ou Serviço
MA1	SCRUM	Acima de 2.000 funcionários	Site de notícias integrado a vários outros portais de informação.
MA2	Extremme Programming	Aproximadamente 30 funcionários	Ambiente de desenvolvimento, entrega e suporte para aplicativos móveis.
MT1	RUP	Acima de 5.000 funcionários	Customização de ERP
MT2	Cascata	Aproximadamente 1.900 funcionários	Aplicativo para CRM.

3.4.1 Empresa MA1

A empresa MA1 trabalha com o métodos ágil SCRUM há onze meses (na época da entrevista). Os desenvolvedores conheciam métodos de desenvolvimento tradicionais e tiveram treinamentos de um a dois meses para migrar de método de desenvolvimento. Foi realizada uma observação direta e análise de alguns documentos da parte de testes.

O desenvolvimento segue as premissas básicas do SCRUM. Todos os dias às 9 horas da manhã é realizado o *daily meeting* com todos os participantes da equipe. A equipe utiliza um quadro com as histórias de usuário escritas em papéis do tipo *post-it*. Esse quadro funciona com um Kanban, cada ticket possui a história de usuário com uma estimativa de tempo e uma figura colada nela denominada *avatar*. O *avatar* é uma personagem qualquer, real ou não, que representa uma pessoa da equipe responsável pelo desenvolvimento daquela história de usuário. É utilizado também um segundo adesivo com a figura de uma joaninha para ilustrar um erro encontrado naquela história de usuário durante algum teste. A Figura 3.1 representa um modelo do Kanban utilizado.

Figura 3.1: Kanban utilizado na empresa MA1

Os casos de teste são escritos juntamente com as histórias de usuário, e disponibilizados em uma planilha que é compartilhada com toda a equipe. A medida que as histórias terminam de serem desenvolvidas a equipe de testes inicia a execução dos casos de teste. Os testes são funcionais e o resultado dos mesmos é compartilhado na mesma planilha dos casos com toda a equipe.

A entrevista foi realizada com uma analista de testes/desenvolvedora e uma analista de negócios que faz o papel de *product owner*. As estimativas são feitas de acordo com

a dificuldade de cada história de usuário. Para classificar as dificuldades são utilizados pontos seguindo a série de fibonacci com limite de pontuação em 21 pontos, ou seja os pontos possível de serem atribuídos são 1, 2, 3, 5, 8, 13, 21. Cada *sprint* entrega um determinado número de pontos que são calculados empiricamente baseado em experiências anteriores.

A quantidade de erros encontrada é utilizada apenas para controle do *sprint* em andamento. O Scrum Master gera um *burn-down chart* com a pontuação do sprint, identificando quantidade de erros encontrados. A métrica é coletada através do preenchimento da planilha de testes, e são utilizadas para debate na reunião de retrospectiva do *sprint*.

O membros da equipe reportaram que sentem falta de automatização de alguns mecanismos como por exemplo ferramentas de teste e análises de métricas mais precisas e instantâneas. A empresa trabalhou com outros métodos de desenvolvimento mais tradicionais e o SCRUM foi muito bem aceito. Depois de um período de transição curto a equipe se acostumou e defende muito o método de desenvolvimento. Para a analista de testes as entregas realmente estão mais rápidas e está mais simples entender e/ou corrigir o escopo, item que era antes um dos maiores problemas encontrados no desenvolvimento e fator de discussão com a área de negócio da empresa. Tanto a analista de negócios quanto a de testes citaram a mudança no ambiente de trabalho. Como o processo como um todo passou a ser transparente a equipe se uniu para cumprir prazos e realizar entregas com qualidade (neste caso se define como satisfação do cliente e baixo número de erros encontrados em produção).

3.4.2 Empresa MA2

A empresa MA2 trabalha com o método ágil XP - *Extremme Programming*, com algumas características aproveitadas do *Lean Software Development*. A empresa era fundamentada em desenvolvimento com método cascata. A mudança para métodos ágeis surgiu de um novo sócio que entrou na empresa para melhorar o problema de qualidade das entregas. Foi realizada uma observação direta e uma entrevista com o sócio mencionado.

A história da transição de um método mais tradicional para métodos ágeis é interessante e pode ser explorada como um estudo de caso a parte. Como essa história aborda assuntos como melhoria de qualidade ela será transcrita brevemente nesta subseção.

A empresa MA2 iniciou a transição para métodos ágeis trabalhando com SCRUM, depois de algum tempo, com conceitos e cultura ágil disseminados na empresa, migrou para o XP. Problemas como dificuldade na manutenção, transferência de conhecimento entre equipes de desenvolvimento e rastreabilidade de erros eram constantes. Com a transição de método, houve uma mudança cultural muito grande. Foi necessário envolvimento de alguns clientes para que eles participassem dessa nova experiência. Além disso, alguns

membros da equipe não se sentiram confortáveis com o novo método e acabaram deixando a equipe. Depois de algumas reuniões de retrospectiva de *sprint* conturbadas, a equipe começou a entender melhor o processo. Em conversa com alguns desenvolvedores foi possível identificar a compatibilidade da equipe com métodos ágeis de desenvolvimento. Quatro desenvolvedores disseram que não se imaginam trabalhando com desenvolvimento sem utilizar métodos ágeis.

A MA2 encontrou uma forma interessante de se disseminar o conhecimento entre a equipe. O foco da empresa é em um produto, porém o mesmo produto possui diversas customizações para diversos clientes, aumentando o problema de manutenção da solução. Para combater esse problema a equipe de desenvolvimento trabalha com um grande *backlog* que não é separado por clientes. Dessa forma, qualquer desenvolvedor terá que em algum momento desenvolver algo para cada cliente da empresa. Caso a equipe seja reduzida, qualquer outro desenvolvedor está apto para assumir as histórias de desenvolvimento.

Na equipe da MA2, todos escrevem e revisam código, seguindo a regra de que quem escreve não revisa. O código é gerado utilizando a técnica de TDD, e os testes são submetidos para um servidor de integração contínua próprio. Todo dia o servidor envia o resultado dos testes para a equipe. A equipe de atendimento mantém métrica de tempo de resposta e quantidade de erros por versão do *software*. Nenhuma versão é lançada com erros conhecidos.

A diferença na qualidade da entrega do software foi percebida, segundo o sócio da empresa, pelos clientes. Antes insatisfeitos, hoje solicitam palestras sobre métodos ágeis para outros fornecedores.

3.4.3 Empresa MT1

A empresa MT1 é fornecedora de serviços de consultoria, tecnologia e terceirização. Está presente em 40 países, com mais de cento e vinte e cinco mil funcionários, possui quase seis mil funcionários diretos no Brasil. A MT1 possui certificação CMMI DEV nível 5.

A entrevista foi realizada com um gerente de projeto e dois desenvolvedores plenos, separadamente. A empresa utiliza o método de desenvolvimento RUP, para criar novos módulos e fazer customização de um ERP. A equipe entrevistada é formada por oito desenvolvedores. Não foi possível realizar uma observação direta, e mesmo com acordo de confidencialidade, nenhum documento de apoio foi fornecido além de material comercial da empresa.

O método de desenvolvimento descrito por todos os entrevistados está em conformidade com o RUP. Toda documentação é gerada por ferramentas da família *Rational Rose*, e são armazenados em uma base de conhecimento utilizada em todos os projetos da empresa. A equipe possui pessoas dedicadas a atividade de teste. Os testes são funcionais

e reportados em documento da ferramenta de produtividade Microsoft Word. Os testes são criados baseados no casos de uso. Nenhuma métrica é gerada com base no resultado dos testes.

A empresa possui uma área chamada PMO Corporativo. É um escritório de projetos que é responsável, entre outras coisas, por fazer controle de qualidade dos projetos da empresa. O foco principal da avaliação é a documentação de gestão de projeto, conferindo se o projeto tem um plano, cronograma, matriz de riscos e apresentações de acompanhamento. É avaliado porém se os documentos gerados pelo projeto estão em conformidade com os modelos disponibilizados pela empresa. A avaliação é realizada semestralmente e a nota é calculada pela PMO Corporativo. A equipe do projeto em questão não tem acesso ou não conhece a forma de pontuação dos projetos.

Os desenvolvedores acreditam que as entregas realizadas possuem um nível bom de qualidade devido ao pequeno número de erros reportados por clientes, entretanto todos citaram a falta de uma avaliação mais completa, com ferramental de apoio. Outro ponto indicado como problemático é o tempo gasto para se atualizar documentação, pois a equipe possui um rotatividade alta de profissionais, o que torna a atualização da documentação um item crítico.

A MT1 trabalha também com outros métodos de desenvolvimento, incluindo métodos ágeis, entretanto a equipe entrevistada não possui experiência com outros métodos.

3.4.4 Empresa MT2

A empresa MT2 atua em quatro áreas diferentes do mercado, sendo elas inovação e inteligência em CRM, gestão de ambientes de tecnologia da informação, soluções de software e engenharia e gestão de serviços e aplicativos. A MT2 possui certificado ISO 9001.

Foi realizada entrevista com um gerente de projeto e um desenvolvedor da solução de CRM da MT2. Não foi possível uma observação direta nem análise de documentos, porém o gerente fez uma visita programada mostrando ferramentas e documentos citados nas entrevistas. A empresa segue conceitos preconizados no PMBoK versão 4. O projeto é sequencial, o desenvolvimento segue cinco fases no modelo cascata (requisitos, planejamento, implementação, testes e manutenção).

A MT2 trabalha com o mesmo método de desenvolvimento desde o seu início, nenhum outro método é utilizado na empresa. Na fase de planejamento, a empresa cria uma especificação funcional. Essa especificação inclui casos de uso, estimativa de desenvolvimento em horas e desenho da arquitetura do software quando necessário (principalmente nos casos onde é necessária integração com outros softwares). O documento é criado por desenvolvedores mais experientes junto ao gerente de projeto.

O acompanhamento do projeto é realizado na ferramenta Microsoft Project onde são utilizados indicadores preconizados no PMBoK como CPI e SPI. A especificação funcional é repassada a equipe de desenvolvimento e de controle de qualidade que inicia a criação de casos de teste baseado nos casos de uso. Os testes são criados utilizando ferramenta da família *unit*. Os erros encontrados podem ser reportados em uma ferramenta interna que serve como base de conhecimento. O ambiente de desenvolvimento faz avaliação automática de padronização de código, impossibilitando o programador de enviar código ao repositório que esteja fora do padrão da empresa. Semanalmente é gerado um relatório com erros encontrados nos testes.

Quando o projeto passa para a fase de manutenção, é trocada a equipe de projeto pela equipe de manutenção. A equipe de manutenção fica em uma sala denominada *war room*. Essa sala possui diversos monitores com métricas em tempo real de quantidade de erros encontrados e não resolvidos, tempo médio para correção de um erro (com indicativo de maior tempo em aberto), reincidência de erro por cliente.

A equipe acredita que a documentação é a grande responsável pelas boas avaliações de satisfação do cliente, pois sem a documentação a equipe de manutenção não teria como realizar um atendimento eficaz e eficiente. O desenvolvedor entretanto relatou que existem alguns problemas de atualização da documentação que nem sempre ocorre como deveria.

3.5 Análise de Dados

As entrevistas foram transcritas e foi realizada uma análise e interpretação dos múltiplos casos em contraste com o referencial teórico através de uma categorização das informações obtidas.

No estudo, foram consideradas como atividades de garantia de qualidade, todas e quaisquer atividades que possuam um padrão sistemático, planejado e que sempre devem ser executadas durante o processo de desenvolvimento, com foco em garantir a qualidade do produto. Todas as empresas entrevistadas apresentaram atividades de garantia de qualidade. A Figura 3.2 sumariza as atividades identificadas para cada empresa.

É possível perceber na figura 3.2 que todas as empresas tem uma preocupação com as atividades de teste de software, com foco no teste funcional. Nos casos das empresas que utilizam métodos tradicionais de desenvolvimento é realizado até teste com o usuário final, executando casos de teste manualmente e reportando os resultados em planilhas. Os métodos ágeis utilizam-se de ferramentas de automatização dos testes. Apesar dos processos de desenvolvimento serem bastante diferentes, cada empresa dentro do seu contexto, realiza um conjunto parecido de atividades de garantia de qualidade, diferenciando-se apenas na forma de divulgação e armazenamento dessas informações. Os métodos ágeis

Figura 3.2: Atividades de garantia de qualidade identificadas nas empresas.

tem foco na entrega rápida e na adaptabilidade de seu código, portanto a informação é importante no momento do desenvolvimento, enquanto os métodos tradicionais possuem foco na antecipação de problemas futuros.

Para auxiliar a identificação da relação entre as práticas de garantia de qualidade, e qualidade final do produto, foi realizada uma análise SWOT para métodos ágeis e outra para métodos tradicionais considerando apenas os dados extraídos no estudo de caso. O resultado é demonstrado nas Tabelas 3.5 e 3.6.

Os processos tradicionais de desenvolvimento criam um ambiente favorável para coleta e análise de dados sobre qualidade. A dificuldade entretanto é selecionar esses dados e identificar uma forma padronizada de se analisar os mesmos. É possível perceber pelo estudo que mesmo com uma quantidade grande de informação, análises de qualidade não são prática comum nas empresas. A grande quantidade de informação também não significa que a informação seja confiável, pois foram relatados casos de documentação desatualizada.

Analisando as Tabelas 3.5 e 3.6 percebe-se que a documentação é ponto crítico para os todos os tipos de processo de desenvolvimento. Nos métodos tradicionais a documentação é vasta, permitindo que sejam realizadas diversas análises sobre o produto gerado. Entretanto foi identificado que a atualização dessa documentação é um ponto fraco nesses processos. Além da grande quantidade de documentos, o foco na evolução do projeto durante sua execução faz com que a equipe deixe de atualizar a documentação. Nos métodos ágeis a documentação também é o um ponto fraco quando se pensa em avaliação de qualidade. A pouca documentação torna difícil a análise e dificulta conclusões sobre os trabalhos desenvolvidos.

Tabela 3.5: Análise SWOT de qualidade para métodos tradicionais

		Objetivo	
Origem do fator	Interno	Ajuda	Atrapalha
		Pontos Fortes	Pontos Fracos
	Externo	<ul style="list-style-type: none"> • Processo identifica pontos onde podem ser realizadas avaliações de qualidade. • Predefinição de escopo e documentação facilitam planejamento de análise de qualidade. 	<ul style="list-style-type: none"> • Apesar das equipes das empresas identificarem a documentação um ponto forte para melhoria de qualidade, os mesmos citaram que a nem sempre a documentação é atualizada de forma correta e que é uma das atividades que mais consomem tempo da equipe de desenvolvimento.
		Oportunidades	Ameaças
		<ul style="list-style-type: none"> • A quantidade de informação armazenada é vasta e permite análises que podem ajudar na melhoria da qualidade de produtos. 	<ul style="list-style-type: none"> • Não foram identificadas ameaças por fatores externos

Além disso, o excesso de documentação gera uma grande oportunidade para os métodos não ágeis de aferir qualidade. Pode ser definido um processo automatizado de avaliação de métricas de qualidade baseado nos documentos gerados. Existem ferramentas de *workflow* no mercado que realizam esse tipo de análise. No caso dos métodos ágeis, a automatização de atividades é ponto de constante preocupação no quesito teste de software. As duas abordagens possuem oportunidades de avaliação automatizada, entretanto o foco dos métodos ágeis são os testes, e dos métodos não ágeis podem ser tanto os testes quanto os documentos de projeto.

3.6 Dificuldades Encontradas

O estudo enfrentou três grandes dificuldades. A primeira delas foi a seleção de empresas participantes. Foi contatado um grande número de empresas que se dispuseram a participar do estudo. Entretanto quando citado o assunto métrica de qualidade muitas empresas, com métodos ágeis ou tradicionais, desistiram da participação. Mesmo nas em-

Tabela 3.6: Análise SWOT de qualidade para métodos ágeis

		Objetivo	
Origem do fator	Interno	Ajuda	Atrapalha
		Pontos Fortes	Pontos Fracos
	Externo	<ul style="list-style-type: none"> • Processos possuem atividades como teste, revisão de código e refatoração já definidos. • Equipes se mostram preocupadas com qualidade do produto, principalmente no ponto satisfação do cliente. 	<ul style="list-style-type: none"> • Pouca documentação limita as possibilidades de análise • Dinâmica de trabalho não permite avaliações de controle de qualidade morosas pois essas podem atrasar ou engessar o processo de desenvolvimento.
		Oportunidades	Ameaças
		<ul style="list-style-type: none"> • Utilização de servidores de integração contínua • Comunicação mais direta com o usuário final. 	<ul style="list-style-type: none"> • Flexibilidade de adaptação somada a pouca documentação dificulta a análise de conformidade a requisitos.

presas participantes e que possuem uma área de garantia de qualidade, equipe de testes é difícil encontrar indicadores ou métricas para avaliação de qualidade dos produtos.

Outra grande dificuldade encontrada foi selecionar empresas que realmente praticam métodos ágeis. Muitas empresas utilizam um artefato, ou uma reunião de algum processo ágil e utilizam esse item como portfolio de vendas do trabalho da empresa.

Por fim, a análise de dados qualitativos para fim de comparação ficou complexa. O fato da entrevista ser semiestruturada facilitou a conversa com empresas, mas dificultou o trabalho de análise dos dados.

3.7 Considerações Finais

O sucesso de qualquer processo de desenvolvimento está diretamente relacionado com o sucesso ou falha do produto final (Hashmi e Baik, 2007b). É importante que todas as pessoas envolvidas em um projeto de desenvolvimento de software saibam os benefícios do processo que está sendo utilizado. A relação entre atividades de garantia de qualidade e a qualidade final de um produto deve ser de tal forma que participantes deveriam ser motivados a seguir fielmente os processos de desenvolvimento. É importante que os desenvolvedores tenham informações sobre a qualidade do software produzido.

O estudo mostrou que as atividades de garantia de qualidade existem desde os estágios iniciais dos métodos ágeis. As entrevistas permitiram concluir também que os praticantes de métodos ágeis são entusiastas do método de desenvolvimento utilizado. Eles tem maior interesse em promover o resultado de seus trabalhos e demonstram total confiança na qualidade de seus produtos. Houve visivelmente, no cenário desse estudo de caso, um comprometimento maior desses desenvolvedores com o processo de desenvolvimento.

Foi possível perceber no estudo de caso que os métodos ágeis possuem práticas de garantia de qualidade. Além disso a frequência dessas práticas, como teste de software e revisão de código, são maiores nos métodos ágeis do que nos métodos tradicionais. É possível identificar resultados semelhantes na literatura. Em Huo et al. (2004b) são comparadas atividades de garantia de qualidade utilizando métodos tradicionais e ágeis. Em outro estudo de caso Hashmi e Baik (2007a) compara os métodos XP e espiral e conclui que o método XP possui atividade de garantia de qualidade que são executadas contínua e repetidamente. Atividades frequentes no processo de desenvolvimento, como planejamento de uma iteração, refatoração e *feedback* de clientes atuam na melhoria de qualidade, identificação e mitigação de riscos dos projetos (Hashmi e Baik, 2007a).

Os praticantes de métodos ágeis entrevistados apontaram a ausência de automação de algumas atividades como por exemplo avaliação de satisfação do cliente. Não existe intenção de gerar histórico ou base de conhecimento, mas sim de obter informações de *feedback* de forma estruturada para facilitar análises em tempo de projeto.

O objetivo do estudo de caso foi investigar as relações de processos de desenvolvimento com atividades de garantia de qualidade. Os modelos de qualidade mais consagrados atendem os métodos tradicionais de desenvolvimento, porém a relação de qualidade dos métodos ágeis ainda não é convincente (Huo et al., 2004a). A questão que paira é se os métodos ágeis possuem rigor suficiente para garantir a qualidade de um software. O entusiasmo e comprometimento dos praticantes de métodos ágeis criou um novo cenário para o projeto de mestrado. O estudo foi realizado com uma quantidade limitada de empresas, tornando quase impossível fazer grandes afirmações sobre a relação de qualidade com métodos ágeis. Entretanto os indícios são de que um fator extremamente importante para a qualidade do produto é o comprometimento da equipe de projeto com o processo de desenvolvimento. O estudo de caso e a literatura mostram que os processos de desenvolvimento ágil possuem atividades de garantia de qualidade. O diferencial pode estar na cultura criada em torno dos processos ágeis.

Para entender melhor essa relação de entre a cultura ágil e qualidade de software, foi realizado um *survey* com praticantes de métodos ágeis.

O *survey* possui uma população maior, podendo, portanto complementar os resultados obtidos nesse estudo. O resultado desse *survey* é apresentado do Capítulo 4.

Survey - Qualidade e Cultura Ágil

4.1 Considerações Iniciais

Desde a aparição dos métodos ágeis, praticantes desses métodos de desenvolvimento afirmam que a utilização dessas técnicas melhoram a qualidade dos produtos de software. Entretanto, uma análise mais detalhada mostra que falta uma técnica para avaliar como processo ágeis atendem os requisitos de qualidade de software (Mnkandla e Dwolatzky, 2006a). Pouco se sabe sobre quais fatores, práticas ou ferramentas possuem impacto na qualidade de um produto desenvolvido através de um método ágil.

Esse *survey* foi motivado por um estudo de caso sobre qualidade de software e processos de desenvolvimento. Acredita-se que um dos fatores fundamentais para a qualidade do produto de software no desenvolvimento ágil seja o comprometimento dos desenvolvedores com o processo. É cultural a diferença de postura dos desenvolvedores perante os desafios e atividades do processo de desenvolvimento. Este *survey* aprofunda a visão do impacto dessa cultura.

A realização de *surveys* é uma estratégia conhecida para se realizar estudos empíricos, muito utilizada em ciências sociais. De acordo com (Punter et al., 2003), um *survey* é uma estratégia para um estudo empírico que fornece uma descrição quantitativa ou numérica sobre a fração de um população - uma amostra - através de questionários (Creswell, 2008; Floyd J Fowler, 2009). Segundo (Pfleeger, 2001), *surveys* podem ter caráter descritivo.

Esse tipo de *survey* geralmente tem como objetivo fornecer uma visão do estado da arte de algum método, ferramenta ou técnica.

O *survey* descrito neste trabalho foi conduzido de acordo com os seguintes passos:

- Definição do estudo - Determinar o objetivo do estudo
- Planejamento do estudo - Operacionalizar os objetivos do estudo em um conjunto de questões e selecionar os participantes.
- Implementação - Operacionalizar o planejamento para que o estudo possa ser realizado.
- Execução - Coleta e processamento de dados.
- Análise - Interpretação dos resultados.

Este capítulo descreve o protocolo e a execução de um *survey* sobre cultura ágil e qualidade de software.

4.2 Objetivo

Foi investigada a literatura existente sobre métodos ágeis considerando comportamento pessoal e foram encontradas diversas pesquisas que mencionam a importância do comprometimento nos métodos ágeis. Não foram encontradas entretanto evidências que apoiem essa afirmação, entretanto existem pesquisas como (Boehm, 2002a; Cockburn e Highsmith, 2001a; Pikkarainen et al., 2008; Ramesh et al., 2006), com diferentes objetivos, que consideram o comprometimento como um fator importante para o sucesso da utilização de um método ágil.

Esse *survey* tenta identificar por que praticantes de métodos ágeis tem uma postura diferente sobre o método de desenvolvimento e qual o impacto desta postura na qualidade do produto resultante.

4.3 Protocolo

O tempo de execução desse *survey* foi determinado para um mês por conveniência. O questionário foi enviado para quatro empresas conhecidas por trabalharem com métodos ágeis. O questionário também foi publicado em redes sociais com foco em métodos ágeis.

Para criar e ajudar o questionário foram escolhidos dois participantes praticantes de métodos ágeis, com experiência de pelo menos um ano. Foi concluído que devido a subjetividade das questões, o *survey* deveria ter foco em algumas questões dissertativas,

para que se possa entender melhor o contexto de trabalho dos praticantes de métodos ágeis.

O questionário foi formulado em cinco partes. A primeira parte é relacionada a informações básicas apenas com intuito de classificar os participantes. A segunda parte aborda o processo de desenvolvimento, como foco em identificar adaptações aos métodos utilizados. A terceira e quarta partes abordam especificamente atividades de garantia de qualidade e teste de software. A motivo das atividades de teste estarem separadas da parte de garantia de qualidade é para dar maior profundidade ao assunto, pois acredita-se que os praticantes de métodos ágeis tenham um bom conhecimento do domínio. A última parte da avaliação são questões abertas sobre o método de desenvolvimento e qualidade de software.

4.3.1 Questões

As questões criadas para o *survey* foram, em sua maioria, de caráter exploratório, com alguns itens parcialmente estruturados conforme descrito em (Miller, 1991; Punter et al., 2003). A Tabela 4.1 apresenta as questões utilizadas no *survey* juntamente com a categoria e classificação da pergunta.

4.3.2 Ameaças a Validade

Ameaças estão presentes em qualquer tipo de investigação. Assim é importante identificá-las apropriadamente. Considerando esse estudo, as seguintes ameaças foram identificadas:

- Alto nível de desistências - Para evitar formulários incompletos, foram criadas seções mais simples no início, fazendo com que o participante se sinta mais confortável e compreenda o contexto do questionário.
- Garantia de conhecimento dos participantes sobre o tema métodos ágeis - Para garantir que os envolvidos no *survey* tenham um conhecimento mínimo sobre métodos ágeis, foram convidadas empresas já conhecidas pela utilização de métodos ágeis, aumentando significativamente a possibilidade dos participantes conhecerem os assuntos abordados.
- Questões abertas - Para mitigar o risco de ter muitas questões abertas, as perguntas foram classificadas por tema. Quando a resposta para a questão não foi clara, ou estava fora do contexto da pergunta, a resposta foi desconsiderada.
- A informação fornecida não é confiável - Se a informação coletada não representa um método ágil, ou não corresponde a pergunta realizada, o formulário foi descartado.

Tabela 4.1: Questões do *survey*

1. Informações básicas		
1.1	Qual método ágil você utiliza com maior frequência ?	Questão estruturada (múltipla escolha)
1.2	Há quanto tempo você utiliza métodos ágeis ?	Questão estruturada (múltipla escolha)
1.3	Você tem experiência com métodos de desenvolvimento tradicionais ? Se sim, quanto tempo ?	Questão estruturada (múltipla escolha)
2. Processo de desenvolvimento		
2.1	Descreva o método de desenvolvimento ágil utilizado ? Ele possui alguma adaptação ?	Questão aberta
2.2	Quais os papéis envolvidos no desenvolvimento ?	Item parcialmente estruturado
2.3	É gerada alguma documentação em relação ao escopo do projeto ?	Questão estruturada (múltipla escolha)
2.4	É gerada alguma documentação em relação ao código ?	Questão estruturada (múltipla escolha)
2.5	É utilizada alguma ferramenta para gestão do projeto ?	Questão estruturada (múltipla escolha)
3. Garantia de qualidade		
3.1	Você realiza alguma atividade de garantia de qualidade (ex. revisão de código, revisão de planejamento, testes) ?	Item parcialmente estruturado
3.2	É gerada alguma métrica referente a qualidade de software ?	Questão estruturada (múltipla escolha)
3.3	A empresa possui equipe dedicada para controle de qualidade ?	Questão estruturada (múltipla escolha)
4. Teste de software		
4.1	Como são gerados os casos de teste ?	Questão aberta
4.2	A empresa possui uma equipe dedicada para a atividade de teste de software ?	Questão estruturada (múltipla escolha)
4.3	É utilizada alguma ferramenta para teste de software ? Se sim, qual ?	Questão aberta
4.4	É realizado algum teste de aceite do usuário final ?	Questão estruturada (múltipla escolha)
4.5	A empresa utiliza algum servidor de integração contínua ? Qual ?	Questão estruturada (múltipla escolha)
5. Considerações finais		
5.1	Qual sua visão sobre satisfação do cliente quando trabalhando com métodos ágeis ?	Questão aberta
5.2	Se você já trabalhou com métodos de desenvolvimento tradicionais, você sente diferença na qualidade dos produtos gerados ? Se sim, por favor descreva-as.	Questão aberta
5.3	Qual sua visão sobre o processo de desenvolvimento utilizado ? Quais pontos fortes e fracos ?	Questão aberta
5.4	Qual sua opinião sobre a qualidade dos softwares gerados utilizando métodos ágeis ?	Questão aberta

- Procedimento do *survey* ou questões não estão claras para os participantes - Foi realizado um piloto para verificar possíveis problemas nas questões. Além disso, o procedimento foi descrito antes do início das perguntas, em forma eletrônica, de modo a esclarecer eventuais dúvidas.

4.4 Coleta de Dados

O *survey* foi realizado *online* entre Maio e Junho de 2013, para facilitar o processo de coleta de informações. Além disso, os participantes puderam responder as questões em seu próprio tempo. O questionário foi formulado utilizando a ferramenta *web Google Forms*¹. Após um mês de execução, o questionário foi fechado e os dados exportados para a ferramenta *Microsoft Excel* para processamento dos dados.

O questionário foi respondido por 28 pessoas as quais são oriundas de seis empresas que foram identificadas. Essas empresas são na sua maioria empresas de pequeno porte. Três dos participantes não identificaram a empresa em que trabalham e coincidentemente os três formulários preenchidos por esses participantes foram descartados. Os motivos do descarte foram, primeiramente, a descrição do método de desenvolvimento que não foi encontrada na literatura e em segundo lugar, a existência de respostas contraditórias.

4.4.1 Análise dos Dados Quantitativos

A distribuição de métodos ágeis entre os participantes no *survey* foi como mostra a Figura 4.1. Pode se observar que a maior parte dos participantes utilizam o método ágil SCRUM. Foram consideradas apenas os formulários que descreveram um método minimamente parecido com o descrito na literatura.

Figura 4.1: Distribuição da utilização dos métodos ágeis entre os participantes do *survey*

¹<http://www.google.com/google-d-s/createforms.html>

Considerando a descrição do processo de desenvolvimento pelos participantes, foi possível perceber um alto nível de concordância entre os métodos descritos e a definição dos métodos na literatura. A Figura 4.2 mostra que mais de 90% dos participantes descreveram a instanciização de um método muito próximo, ou exatamente igual a definição original do processo.

Figura 4.2: Classificação da descrição dos métodos utilizados pelos participantes em comparação com a literatura.

A Figura 4.3 mostra algumas atividades de garantia de qualidade citadas pelos desenvolvedores. Apesar de muitas respostas enaltecerem a importância dos testes, a atividade de revisão de código foi a que mais se destacou entre as respostas obtidas, sendo executada por quase todos os participantes. Em alguns casos os participantes citaram que a utilização de programação pareada deveria ser considerada como atividade de revisão de código, entretanto os pesquisadores não possuem evidências suficientes para concordar com tal afirmação.

Figura 4.3: Atividades realizadas pelos participantes a cada iteração de desenvolvimento

A Figura 4.4 mostra que mais de um terço dos participantes, trabalharam ao lado de uma equipe dedicada à atividade de teste de software. Alguns participantes citaram

que a equipe de desenvolvimento possuía pelo menos um profissional alocado somente na atividade de teste de software, mas ao final de uma iteração a equipe inteira realiza testes. Mesmo nesses casos o controle das atividades de teste e atualização de documentação era realizado por este profissional especializado.

Figura 4.4: Distribuição dos participantes de acordo com participação em equipes de desenvolvimento que possuem equipe dedicada a atividade de testes.

Considerando a importância dada pelos próprios desenvolvedores a atividade de testes, a Figura 4.5 identifica quais as ferramentas de testes que são mais utilizadas. É possível perceber o foco no teste funcional, e a Figura 4.6 colabora com essa informação indicando alto nível de testes baseado na história de usuário. Corroborar ainda para essa afirmação a grande utilização da ferramenta *Selenium*², que simula um usuário realizando a atividade de testes em uma interface web.

Figura 4.5: Utilização de ferramentas de apoio a atividade de teste de software pelos participantes.

A Figura 4.6 identifica ainda a utilização de ferramentas mais completas para a atividade de teste de software. É o caso da EMMA³. A EMMA é um conjunto de ferramentas

²<http://docs.seleniumhq.org/>

³<http://emma.sourceforge.net/>

de código aberto para medição e elaboração de relatórios sobre a cobertura de casos de testes em códigos criados na linguagem Java.

Figura 4.6: Forma com que as equipes dos participantes geram os casos de teste para cada iteração.

4.4.2 Análise dos Resultados Qualitativos

Além dos dados quantitativos foi possível extrair algumas informações das respostas em questões abertas, que colaboram com a visão de que a cultura ágil faz com que o desenvolvedor se preocupe com o seu processo de desenvolvimento.

Dentre as respostas podemos identificar que muitos desenvolvedores acreditam que a utilização do método ágil auxilia no controle e monitoração do projeto. Além disso, o fato da equipe inteira possuir todas as informações disponíveis sobre o projeto ajuda a entender o escopo e aceitar as mudanças. Outro ponto levantado foi de que o foco em pequenas funcionalidades facilita o desenvolvimento e manutenção.

Alguns desenvolvedores não imaginam trabalhar com outros métodos que não tenham os mesmos valores que os métodos ágeis. Essa afirmação é interessante, considerando que não havia uma questão específica sobre mudança de método de desenvolvimento de métodos ágil para algum outro.

Após compreender a dinâmica do método ágil utilizado, muitos afirmaram que mudaram a postura perante desenvolvimento de software. Mais da metade dos desenvolvedores afirmaram ficar animados quando se deparam com projetos desafiadores, citando que a dinâmica do trabalho mantém todos motivados. Atividades como *daily meeting* do SCRUM motivou desenvolvedores a melhorar suas habilidades de comunicação.

Poucos desenvolvedores indicaram contato constante com clientes e/ou *stakeholders*, o que é contraditório com o que se diz no manifesto ágil. Alguns informaram ainda que após transição para métodos ágeis, alguns funcionários decidiram deixar a empresa pois não se adaptaram ao novo perfil de trabalho.

4.5 Considerações Finais

Os resultados colaboram para a suspeita inicial de que existe um entusiasmo ou uma identificação muito grande entre desenvolvedores e métodos ágeis. Os participantes demonstraram conhecimento do processo utilizado e identificaram em suas respostas a importância de atividades de garantia de qualidade em especial teste de software.

Apesar do baixo nível de documentação, foi indicado que as equipes possuem meios de comunicação eficientes e as informações sobre o projeto são propagadas de forma natural. É, novamente, possível perceber nas respostas que praticantes de métodos ágeis estão sempre querendo demonstrar o resultado de seu trabalho, conforme indicado por (Mnkandla e Dwolatzky, 2006b).

Os praticantes de métodos ágeis acreditam que o processo que eles utilizam é uma das melhores formas de se criar um produto de qualidade. A cultura ágil muda a postura dos agentes do processo e indica que pode ser uma das principais razões pela qual métodos ágeis criam produtos de software com alta qualidade.

Dos 28 formulários obtidos apenas 3 foram descartados, todos de participantes provenientes do convite realizado na rede social LinkedIn. Das empresas convidadas 21 pessoas participaram do *survey*. Os outros quatro participantes são oriundos da rede social LinkedIn, e apenas dois indicaram a empresa para qual trabalham. O número de formulários descartado foi pequeno, o que pode indicar que os participantes conheciam o domínio que foi tratado no *survey*, entretanto mais da metade das empresas onde os participantes trabalham foram convidadas a participar do experimento, e essas empresas são conhecidas no mercado pela utilização de métodos ágeis. É possível que a cultura ágil seja bastante difundida nessas empresas, mais do que a média do mercado. Para avaliar o impacto da participação dessas empresas, é necessário que o *survey* seja conduzido novamente, de preferência sem um foco em empresas, apenas em comunidades ágeis, ampliando o universo de respostas e permitindo uma melhor generalização dos resultados.

IQA - Uma ferramenta de avaliação de qualidade

5.1 Considerações Iniciais

Durante os estudos realizados deste projeto foi identificado um assunto comum entre muitos praticantes de métodos ágeis. Muitos desenvolvedores citaram a necessidade de uma ferramenta automatizada que reunisse algumas informações sobre qualidade dos produtos, com foco em testes e satisfação de desenvolvedores e clientes. Essa afirmação é fortalecida pelos resultados do trabalho realizado por Azizyan et al. (2011). Nele é reportado a condução de um *survey* sobre utilização de ferramentas nos métodos ágeis e um dos pontos mais citados pelos sujeitos da pesquisa é a necessidade de melhores ferramenta para reporte de dados. Os processos de avaliação de qualidade, em sua maioria, são morosos e podem descaracterizar a natureza ágil dos métodos ágeis. A utilização de uma ferramenta que reuna informações básicas sobre o desenvolvimento pode colaborar para a coleta de informações fazendo com que a equipe de projeto possa focar apenas na análise das informações.

Outro fator citado foi que em empresas de prestação de serviço, nem sempre é possível ter o cliente disponível o tempo todo. Muitas vezes algum analista faz o papel de *product owner* para garantir que alguém com conhecimento sobre o produto esteja disponível

o tempo todo durante o desenvolvimento. Uma ferramenta web poderia possivelmente ajudar na aproximação de usuários finais com as equipes de desenvolvimento.

Em (Abbas, 2007) a autora conduziu um estudo empírico sobre qualidade de software no processo de desenvolvimento ágil. Para isso foi construído um monitor de iterações que coletava informações através de formulários online para avaliar a qualidade do processo de desenvolvimento. O monitor porém possui foco na avaliação do projeto de pesquisa e não foi construído com o intuito de fazer uma análise contínua dos dados.

Considerando esse cenário, foi construída uma ferramenta web para auxiliar avaliação de qualidade de projetos que utilizem método ágeis denominada IQA (Iteration Quality Assessment). Essa ferramenta permite concentrar os resultados de avaliação da equipe de desenvolvimento e clientes em um único ambiente, possibilitando uma avaliação do produto durante o desenvolvimento.

O IQA pode ser utilizado tanto por pequenas equipes que pretendem identificar problemas no processo de desenvolvimento que possam afetar a qualidade do produto, bem como por pesquisadores que desejam entender melhor o relacionamento entre qualidade de software e desenvolvimento iterativo.

5.2 Desenvolvimento da Aplicação

O objetivo da ferramenta denominada IQA (*Iteration Quality Assessment*) é manter online formulários simples que auxiliem na avaliação de qualidade de projetos ágeis. A avaliação pode ser realizada por qualquer participante da equipe, desde o desenvolvimento até o cliente final.

5.2.1 Implementação

Para desenvolver o IQA foi escolhida a linguagem de programação PHP com o gerenciador de base de dados PostgreSQL e servidor web Apache. O *framework* utilizado foi o Zend *Framework* v1.12¹, doravante denominado Zend, devido ao conhecimento prévio dos pesquisadores na tecnologia.

O Zend é um *framework* de aplicações web para PHP 5, orientado a objetos, com código fonte aberto, comumente chamado de biblioteca de componentes devido ao seu baixo acoplamento. Apesar do baixo acoplamento, o Zend utiliza o padrão de desenvolvimento MVC (Model-View-Controller). O padrão MVC é uma arquitetura de aplicativos web. Grande parte dos aplicativos web existentes podem ser categorizadas em: apresentação,

¹<http://framework.zend.com/>

lógica de negócios e modelo de dados. Na arquitetura MVC essa separação é bastante clara e bem definida, conforme mostra a Figura 5.1:

Figura 5.1: Padrão de Desenvolvimento MVC (Model-View-Controller)

- Modelo de Dados - É a parte da aplicação que define suas funcionalidades básicas atrás de um conjunto de abstrações. Nessa camada são definidas rotinas de acesso de dados e algumas lógicas de negócio.
- Apresentação - Essa camada define o que será apresentado para o usuário. Geralmente o controlador envia os dados para esta camada que apresenta os dados para o usuário em algum formato. Essas camadas também coletam informações dos usuários. Em aplicativos web a tecnologia mais utilizada para a camada de apresentação é o HTML.
- Controlador - O controlador conecta todas as camadas. Ele manipula os modelos de dados, decide qual apresentação será utilizada baseado em ações do usuário e outros fatores, envia os dados para a apresentação quando necessário.

A Figura 5.2 mostra a tela de questionário da ferramenta implementada. A aplicação foi desenvolvida em três meses, de Julho à Setembro de 2013. Após a implementação, a ferramenta foi validada por meio do seu uso em uma empresa real. Essa validação permitiu fazer os ajustes necessários na ferramenta.

5.2.2 Questões

A ferramenta é composta por um conjunto de questões que buscam identificar atividades realizadas durante uma iteração e coletar *feedback* de desenvolvedores e clientes sobre o desenvolvimento do software, quantificando essas informações para auxiliar análises de controle de qualidade.

Figura 5.2: Tela da Ferramenta IQA

As perguntas criadas foram direcionadas ao mapeamento dos tipos de atividades realizadas, se houveram dificuldades no desenvolvimento da iteração e foram divididas em cinco seções.

- Avaliação da iteração - Perguntas sobre a interação com a equipe, *stakeholders* e clientes, tempo da iteração, planejamento e documentação.
- Atividades realizadas pelo participante - Indicação de quais tipos de atividades foram realizadas pelo participante (e.g. Refatoração, revisão de documentos, desenvolvimento, testes).
- Comunicação - Mapeamento sobre como são realizadas as comunicações em uma iteração.
- Foco - Indicação de qual foi o foco das atividades do participante na iteração.
- Observações - Questão aberta para que o participante possa registrar alguma observação sobre a iteração.

O número de perguntas é pequeno para evitar que a avaliação tire o foco das atividades de desenvolvimento. Exceto pela ultima seção de observações, todas as outras perguntas são de múltipla escolha. Para realizar a análise das respostas, a escala Likert (Likert, 1932) foi utilizada, atribuindo valores quantitativos a dados qualitativos. A escala Likert é um tipo de escala de resposta psicométrica usada habitualmente em questionários. Ao responderem a um questionário baseado nesta escala, os participantes especificam seu nível de concordância com uma afirmação (Likert, 1932).

Os valores utilizadas nas possíveis respostas do questionário foram:

- 1 - Não concordo totalmente
- 2 - Não concordo parcialmente
- 3 - Indiferente
- 4 - Concordo parcialmente
- 5 - Concordo totalmente

5.3 Uso da Ferramenta em um Ambiente Real

A empresa que participou do piloto utiliza o SCRUM como método ágil de desenvolvimento. Para proteger a empresa participante e sua equipe de projeto, os nomes foram ocultados neste documento. Cada profissional foi chamado pelo seu papel na equipe de desenvolvimento e a empresa foi chamada pelo nome Empresa A. A Empresa A possui 14 funcionários e atua no ramo de serviços com foco no mercado financeiro e de telecomunicações.

A equipe de projeto possui 5 pessoas entre *product owner*, *SCRUM master* e três desenvolvedores. A ferramenta foi utilizada durante dois *sprints* de desenvolvimento coletando *feedback* dos envolvidos no processo de desenvolvimento. O projeto em desenvolvimento era um ERP. Na época foram identificados problemas de requisitos no projeto. Do lado do cliente existiam três clientes como ponto de comunicação, o que dificultou a definição dos requisitos.

Para utilização foram configurados duas iterações de 15 dias. A data para encerramento dos questionários era até 2 dias depois do encerramento de cada iteração, e o questionário poderia ser respondido a qualquer momento durante a iteração. Os resultados só foram publicados após encerramento do prazo de resposta, e a questão dissertativa foi aberta a todos os participantes da iteração, para evitar que a ferramenta criasse uma barreira entre os integrantes da equipe de projeto.

5.4 Coleta de Dados

A coleta de dados foi realizada entre Outubro e Novembro de 2013. Neste período a ferramenta foi implantada na empresa e a equipe de projeto teve um treinamento sobre a utilização do IQA, oportunidade onde puderam tirar dúvidas sobre as questões.

Durante as duas iterações realizadas a equipe de projeto respondeu o questionário antes do final da iteração. Já com os clientes foi necessário fazer operação assistida para que eles respondessem os questionários. A operação assistida foi realizada pelo *product owner* após o encerramento da iteração.

5.5 Análise de Dados

Dentre as respostas dos questionários podemos destacar que os desenvolvedores:

- acreditam que possuem um bom relacionamento com a equipe de projeto,
- indicaram estar motivados para solucionar os problemas do projeto,
- acharam que as reuniões de início da iteração (*sprint planning*) não foram muito eficazes,
- não houve tempo suficiente para resolver os defeitos encontrados,
- disseram que a reunião de retrospectiva foi bastante efetiva.

Considerando ainda as respostas dos questionários podemos destacar que os clientes:

- possuem uma boa visão do andamento do projeto e dos problemas enfrentados,
- confiam na equipe de desenvolvimento para solucionar os problemas encontrados,
- sentem falta de relatórios menos técnicos do que a ferramenta de gestão utilizada pela Empresa A ².

A ferramenta consolida automaticamente as respostas e cria gráficos para auxiliar a equipe a identificar eventuais problemas. A Figura 5.3 por exemplo mostra o foco da iteração, na visão de cada integrante da equipe para a primeira iteração. É possível perceber uma distancia entre a visão dos desenvolvedores e clientes sobre o que estava sendo construído no *sprint* em questão. Além disso existe uma discordância de foco dentro da própria equipe de desenvolvimento, que é uma equipe pequena.

Para entender esses números uma conversa foi agendada entre pesquisador e equipe. Foi identificado que, apesar de cada pessoa trabalhar em uma parte do sistema, fazendo com que a diferença de visão sobre o foco da iteração possa ser comum, um dos desenvolvedores classificou a refatoração de uma parte de sistema como correção de defeito, pois o motivo da refatoração foi um problema de usabilidade apontado pelo cliente. O *product owner* conduziu uma conversa com os clientes para entender a diferença de visão sobre o foco da iteração e identificou que um dos clientes, ou seja 33%, identificaram as alterações nas tela para usabilidade como defeito, já os outros dois clientes não classificaram essas mudanças como defeito, mas sim como melhorias.

Na segunda iteração a diferença de percepção sobre o foco melhorou em relação ao cliente, mas internamente ainda existiam divergências. Um ponto identificado pelo *SCRUM*

²A Empresa A utiliza a ferramenta de gestão Redmine para criação de suas histórias de usuário, controle de horas e acompanhamento de evolução do projeto.

Figura 5.3: Gráficos gerados pela ferramenta IQA. A Figura 5.3a mostra a distribuição de opinião dos desenvolvedores sobre qual o foco da iteração realizada. A Figura 5.3b mostra a visão do cliente sobre o foco da mesma iteração.

master foi a utilização de ferramentas de comunicação interna. A Figura 5.4 mostra que, apesar da equipe de desenvolvimento estar no mesmo ambiente, incluindo *product owner* e *SCRUM master*, a equipe utiliza constantemente e-mail e ferramentas de mensagem instantânea ³.

Figura 5.4: Gráfico informativo gerado pela IQA sobre frequência de utilização de canais de comunicação.

Foram atribuídos pesos as possíveis respostas para utilização de meios de comunicação no seguinte modelo:

- Diariamente - 5 pontos
- Alta (mais de 5 vezes em uma iteração) - 4 pontos
- Normal (3 a 5 vezes na iteração) - 3 pontos

³No caso da Empresa A é utilizado a ferramenta Google Talk

- Baixa utilização (1 ou 2 vezes na iteração) - 2 pontos
- Não utiliza - 1 ponto

A pontuação final é calculada através de uma média simples das respostas. Os canais de comunicação e-mail e ferramentas de mensagem instantânea tiveram médias 3,4 e 3,8 respectivamente. Esse pode ser um indicativo de que é necessária uma melhoria no processo interno, aumentando o foco na comunicação direta ao invés de utilizar ferramentas para esta atividade.

5.6 Considerações Finais

Este capítulo apresentou o aplicativo web IQA, uma ferramenta para auxiliar equipes ágeis a identificar e compreender problemas durante o processo de desenvolvimento com intuito de auxiliar nas atividades de garantia de qualidade. O IQA pode ser utilizado em situações onde problemas relacionados a qualidade do produto já foi detectada ou durante uma série de iterações para auxiliar as equipes ágeis a controlar a qualidade de seus produtos.

Os dados coletados até o presente momento não são representativos, porém indicam boas perspectivas de uso. O protocolo e execução do experimento devem ser expandidos para que se possa generalizar os dados coletados. A nova execução deve atender mais empresas de diferentes segmentos do mercado e deve ser utilizada durante um maior número de iterações.

A ferramenta ainda necessita de uma integração com ferramentas de teste de software para que a análise dos dados coletados seja completa.

Outra aplicação da IQA é em projetos de pesquisa. A ferramenta demonstrou que pode auxiliar projetos de pesquisa, identificando outros tópicos relacionados ao desenvolvimento de software iterativo.

Conclusão

6.1 Caracterização da Pesquisa Realizada

Este projeto de mestrado teve como principal motivação estudar e analisar o tema qualidade de software no contexto ágil. Métodos ágeis estão sendo bastante empregados pela indústria, porém, pode-se observar que a forma de avaliar a qualidade nesse processo não pode se o mesmo empregado em outros métodos tradicionais, como cascata e processo unificado.

Um estudo de caso foi realizado para tentar entender as diferenças entre métodos mais tradicionais de desenvolvimento e métodos ágeis no relacionamento dos mesmos com atividades de controle e garantia de qualidade. O trabalho identificou que a maior distância entre os dois processos está na documentação. A documentação de um projeto possibilita uma série de análises, e como os métodos ágeis não tem foco na criação de documentos certas análises se tornam mais complexas ou trabalhosas. O estudo apurou que existe pouco conhecimento das empresas sobre conceitos de teste de software que não sejam teste funcional. Os modelos de qualidade conceituados neste trabalho são conhecidos, porém poucos conseguem se aprofundar no tema. Este trabalho empírico demonstrou também um resultado não esperado. Praticantes de métodos ágeis possuem um comportamento peculiar em relação ao processo de desenvolvimento que utilizam. Eles conhecem muito bem o processo e tem uma preocupação em mostrar que o resultado do trabalho possui alto nível de qualidade. Existe uma cultura entre os praticantes de

métodos ágeis que faz com que os processos sejam amplamente divulgados entre as equipes e seguidos com alto grau de concordância com a descrição desses processos na literatura.

Considerando os resultados do estudo de caso, foi realizado também *survey* para investigar o comprometimento dos praticantes de métodos ágeis com as práticas do processo utilizado. Os resultados mostram que o nível de comprometimento dos praticantes de métodos ágeis com os seus métodos de desenvolvimento é alto. A cultura ágil dissemina suas diretrizes entre as pessoas envolvidas fazendo com que as mesmas identifiquem a importância das atividades realizadas. A importância de seguir os preceitos da cultura ágil, de se realizar testes e de dar importância a comunicação tanto interna quanto externa à equipe é ponto fundamental na qualidade dos produtos construídos com métodos ágeis.

A aplicação web IQA, construída neste projeto, é uma ferramenta para avaliação de qualidade baseada em *feedbacks* da equipe de desenvolvimento, clientes e *stakeholders*. Apesar de existir um receio por parte de praticantes de métodos ágeis quanto a utilização de muitas ferramentas, pois as mesmas podem causar distanciamento das equipes e desestimular a comunicação cara-a-cara, a estruturação de algumas informações simples ajudaram os programadores a identificar problemas de escopo e comunicação com o cliente. Muitos desenvolvedores apontam como necessidade ferramentas que forneçam um reporte melhor do andamento e problemas de projeto (Azizyan et al., 2011). A ferramenta necessita de integração com ferramentas de teste para que a análise dos dados coletados seja mais completa.

6.2 Fatores que contribuem para qualidade do produto

O objetivo deste projeto inicial era entender o relacionamento entre alguns processos de desenvolvimento ágil e a qualidade do softwares criados utilizando esses métodos. Inicialmente, pretendia-se propor um *checklist* de atividades de garantia de qualidade, junto com métricas que aferissem qualidade do produto, e que fizessem sentido no contexto do desenvolvimento ágil, dada a falta de diretrizes para esses tópicos nesses métodos.

O estudo de caso e o *survey* realizados indicaram que existem sim fatores nos métodos ágeis que contribuem para a qualidade do software. São eles:

- Atividades de garantia de qualidade bem definidas nos processos.
- Execução de atividades de teste e refatoração com alta frequência.
- Conhecimento do processo de desenvolvimento por parte dos desenvolvedores.
- Importância dada a qualidade de software pelos desenvolvedores.

- Comprometimento com o processo utilizado. Execução das atividades do processo de acordo com a literatura.

O principal diferença encontrada nos métodos ágeis foi a cultura ágil. Os três últimos itens da lista de fatores que contribuem para qualidade do produto no desenvolvimento com métodos ágeis indicam uma diferença comportamental dos desenvolvedores perante os desafios de desenvolvimento de software.

Os métodos ágeis focam em talentos e habilidades das pessoas envolvidas no projeto. Os processos são ajustados para se adequarem a especificidades das equipes de desenvolvimento. Equipes ágeis praticam o liderança por colaboração ao invés de gestão por comando, eles definem metas e restrições, fornecendo limites onde a inovação pode agir. Eles são macro-gestores ao invés de micro-gestores. Eles entendem que quem toma a decisão não é tão importante quanto a colaboração para criação da informação que direciona decisões informadas. Esses fatores influenciam na cultura da equipe que se dispõe a trabalhar com métodos ágeis criando ecossistemas ágeis. As pessoas, o ambiente, a cultura, todos influenciam uns aos outros. Quando a equipe é separada de ambiente físico, por exemplo, a dinâmica de comunicação muda. Um projeto ágil é um ecossistema (Cockburn e Highsmith, 2001a), onde processos, pessoas e ambiente interagem de forma dinâmica, com foco na entrega de um produto e na satisfação do cliente. Os projetos ágeis vem mostrando eficiência em projetos exploratórios, complexos, colaborativos e com foco nas pessoas, criando um ambiente atrativo e motivador para desenvolvedores (Boehm, 2002b; Cockburn e Highsmith, 2001a).

6.3 Contribuições

Dentre as principais contribuições deste trabalho destacam-se:

- A investigação do relacionamento entre processos de desenvolvimento e as práticas de garantia de qualidade de software e teste de software.
- Um protocolo de um estudo de caso sobre visão de equipes de desenvolvimento sobre o seu processo de desenvolvimento, atividades de garantia de qualidade e teste de software.
- Um protocolo de um *survey* sobre conhecimento sobre métodos ágeis e atividades de garantia de qualidade e teste de software.
- Uma ferramenta web para avaliação contínua de *feedbacks* sobre qualidade do produto, foco do desenvolvimento e dificuldades encontradas.

Essas contribuições forneceram dados para a escrita dos seguintes artigos:

- **Artigo enviado ao *The 26th International Conference on Software Engineering and Knowledge Engineering (SEKE 2014)* :** *The Agile Quality Culture: A survey on agile culture and software quality.*
- **Artigo a ser enviado ao *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2014)* :** *Agile and Traditional Quality Assurance : A case study on brazilian companies.*
- **Artigo em elaboração sobre utilização da ferramenta IQA para análise de *feedback* de clientes em empresas de serviço (a ser submetido).**

Além disso, os dados completos do estudo de caso, incluindo seu protocolo, resultados e transcrições das entrevistas serão publicados como Notas do ICMC-USP, Série Computação:

- **Notas do ICMC-USP. Série Computação:** *Estudo de caso - Garantia de Qualidade Ágil e Tradicional.*
- **Notas do ICMC-USP. Série Computação:** *Survey - Qualidade e Cultura Ágil.*

6.4 Dificuldades e Limitações

Durante o desenvolvimento do trabalho foram encontradas dificuldades na interação com as empresas participantes. O contato inicial sempre foi muito positivo, entretanto ao longo do trabalho algumas empresas desistiram de participar do projeto. O motivo mais comum foi a identificação de que a área de desenvolvimento não possuía nenhuma ou quase nenhuma atividade de garantia de qualidade. Mesmo os dados de cada empresa sendo considerados sigilosos, muitas empresas decidiram não participar. Outro ponto crítico foi o tamanho das empresas. Quanto maior a empresa maior a dificuldade de se comunicar internamente. Em dois casos o diretor da área de desenvolvimento autorizou o estudo, entretanto o gerente responsável não colaborou com as informações necessárias.

Um fator limitante do estudo de caso e do *survey* foram a população. Não é possível generalizar os resultados com os tamanhos das amostras consideradas. Entretanto os estudos indicam uma tendência que serve como base para novas investigações.

6.5 Trabalhos Futuros

A realização deste trabalho possibilitou identificar diferentes e importantes direções para trabalhos futuros a serem desenvolvidos sobre qualidade de software nos desenvolvimento com métodos ágeis. Dentre eles, destacam-se:

- Ajuste e nova execução do estudo de caso com amostra maior - As observações diretas proporcionaram uma quantidade de informações muito rica. Pequenos ajustes nas questões, um maior foco nas observações diretas ao invés das entrevistas e uma amostra maior devem fornecer uma visão mais completa do cenário brasileiro de práticas de garantia de qualidade de software, facilitando a generalização dos resultados. Outra possibilidade é dividir as empresas por segmento de mercado, pois as entrevistas realizadas indicaram que a atividade de testes pode variar de acordo com o segmento.
- Aprofundar os estudos sobre motivação das equipes de desenvolvimento e qualidade final do produto - Planejar e executar um mapeamento sistemático sobre motivação dos desenvolvedores e qualidade de software. Criar um estudo de caso para avaliar impactos da motivação dos desenvolvedores na satisfação final do cliente.
- Ampliar o aplicativo IQA - Criar integração com os relatórios disponibilizados pelas ferramentas Unit, Selenium HQ e com dados do servidor de integração contínua Hudson.

Referências

- Abbas, N. Agile software assurance: An empirical study. In: *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, p. 499.
- Abbas, N. *Software Quality and Governance in Agile Software Development*. Tese de Doutoramento, University of Southampton - Faculty of Engineering, Science, and Mathematics School of Electronics and Computer Science, 2009.
- Abrahamsson, P.; Salo, O.; Ronkainen, J.; Warsta, J. Agile Software Development Methods. *VTT technical Research Centre of Finland*, 2002.
- Abran, A.; Moore, J. W. *Guide to the Software Engineering Body of Knowledge*. Angela Burgess, 2004.
- Ambler, S. Quality in an Agile World. *Software Quality Professional* 7, v. 4, p. 34–40, 2005.
- Azizyan, G.; Magarian, M. K.; Kajko-Matsson, M. Survey of Agile Tool Usage and Needs. *2011 AGILE Conference*, p. 29–38, 2011.
- Basili, V. R.; Caldiera, G.; Rombach, H. D. The Goal Question Metric Approach. 1994.
- Beck, K. Embracing Change With Extreme Programming. *Computer*, v. 10, n. 32, p. 70–77, 1999a.
- Beck, K. *Extreme Programming Explained: Embracing Change*. Addison Wesley, 1999b.
- Beck, K.; Andres, C. *Extreme Programming Explained: Engrace Change*. 2nd ed. Addison-Wesley Professional, 2004.

- Beck, K.; Fowler, M. *Planning Extreme Programming*. 1 ed. Boston, MA, USA: Addison Wesley, 139 p., 2001.
- Boehm, B. Get Ready for Agile Methods, with Care. *Computer*, v. 1, p. 64–69, 2002a.
- Boehm, B. Get ready for agile methods, with care. *Computer*, v. 1, p. 64–69, 2002b.
- Boehm, B.; Turner, R. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co. Inc., 2003.
- Boehm, B. W.; Brown, J. R.; Kaspar, H.; Lipow, M.; Macleod, G. J.; Merrit, M. J. *Characteristics of Software Quality*. North-Holland, 1978.
- Brooks, F. P. No silver bullet: essences and accidents of Software Engineering. *Computer*, v. 20, n. 4, p. 10–19, 1987.
- Budd, T. A. Mutations analysis: Ideas, examples and prospects. *Computer Program Testing North-Holand Publishing Company*, p. 129–148, 1981.
- Carosia, J. S. *Levantamento da qualidade do processo de software com foco em pequenas organizações*. Dissertação de Mestrado, INPE - Instituto Nacional de Pesquisas Espaciais, 2004.
- Chulani, S.; Ray, B.; Santhanam, P.; Leszkowicz, R. Metrics for Managing Customer View of Software Quality. In: *Proceedings of the 9th International Symposium on Software Metrics*, Washington, DC, USA: IEEE Computer Society, 2003.
- Cockburn, A. *Agile Software Development*. 1 ed. Boston, MA, USA: Addison Wesley, 278p p., 2002.
- Cockburn, A. *Crystal Clear A Human Powered Methodology for Small Teams*. Addison Wesley, 2005.
- Cockburn, A.; Highsmith, J. Agile Development: The people factor. *Computer*, v. 11, n. 34, p. 131–133, 2001a.
- Cockburn, A.; Highsmith, J. Agile Software Development: The business of innovation. *Computer*, v. 9, n. 34, p. 120–127, 2001b.
- Cockburn, A.; Highsmith, J.; Beck, K.; Jeffries, R.; Beedle, M.; van Bennekum, A.; Cunningham, W.; Fowler, M.; Greening, J.; Hunt, A.; Kern, J.; Merick, B.; Martin, R. C.; Mellor, S.; Schwaber, K.; Sutherland, J.; Thomas, D. Agile Manifesto, <http://agilemanifesto.org/>. 2001.

-
- Cohen, D.; Lindvall, M.; Costa, P. An Introduction to Agile Methods. *Advances in Computers*, , n. 1-66, 2004.
- Creswell, J. W. *Research design: Qualitative, quantitative, and mixed methods approaches*. SAGE Publications, Inc, 2008.
- Crosby, P. B. *Quality is free*. New York: McGraw-Hill, 1992.
- Delamaro, M.; Maldonado, J.; Jino, M. *Introdução ao teste de software*. Editora Campos, 2007.
- DeMillo, R. A.; Lipton, R. J.; Sayward, F. G. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, v. 11, n. 4, p. 34–43, 1978.
- Deutsch, M. S.; Willis, R. R. *Software Quality Engineering, A Total Technical Management Approach*. Prentice Hall, 1988.
- Dromey, R. G. Cornering the Chimera. *IEEE Software*, v. 13, n. 1, p. 33–43, 1996.
- Dyba, T.; Dingsoyr, T. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, v. 50, p. 833–859, 2008.
- Evans, M. W.; Marciniak, J. J. *Software Quality Assurance and Management*. John Wiley and Sons, 1987.
- Floyd J Fowler, J. *Survey research methods*. SAGE Publications, Inc, 216 p., 2009.
- Fowler, M. The New Methodology, <http://www.martinfowler.com/articles/newMethodology.html>. 2005.
- Galin, D. *Software Quality Assurance: From Theory to Implementation*. Addison Wesley, 2003.
- Gillies, A. C. *Software Quality: Theory and Management*. London: Chapman & Hall, 2002a.
- Gillies, A. C. *Software quality: Theory and management*. London: Chapman & Hall, 2002b.
- Harrold, M. J. Testing: A roadmap. In: *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, New York, NY, USA: ACM Press, 2000, p. 61–72.

- Hashmi, S. I.; Baik, J. Software Quality Assurance in XP and Spiral - A Comparative Study. *Proceedings of the The 2007 International Conference Computational Science and its Applications, IEEE Computer Society*, 2007a.
- Hashmi, S. I.; Baik, J. Software quality assurance in xp and spiral - a comparative study. *Proceedings of the The 2007 International Conference Computational Science and its Applications, IEEE Computer Society*, 2007b.
- Highsmith, J. *Adaptive Software Development: a Collaborative Approach to Managing Complex Systems*. Dorset House Publishing Co., 2000.
- Humphrey, W. S. *Managing for innovation: leading technical people*. Prentice Hall, 1987.
- Huo, M.; June, V.; Zhu, L.; Babar, M. A. Software quality and agile methods. In: *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, 28th Annual International Software and Application Conference (COMPSA'04), 2004a.
- Huo, M.; Verner, H.; l Zhu; Babar, M. A., eds. *Software Quality and Agile Methods*, 28th Annual International Software and Application Conference (COMPSA'04), 2004b.
- IEEE {IEEE} Std 610.12-1990 – {IEEE} Standard Glossary of Software Engineering Terminology. In: *IEEE Software Engineering Standards Collection*, New York: The Institute of Electrical and Electronics Engineers, 1991.
- Jeffries, R.; Anderson, A.; Hendrickson, C. *Extreme Programming installed*. 1 ed. Boston, MA, USA: Addison Wesley, 265p p., 2001.
- Juran, J. M.; Feo, J. A. D. *Juran's quality handbook, sixth edition: The complete guide to performance excellence*. Mcgraw-Hill, 2010.
- Kan, S. H. *Metrics and Models in Software Quality Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co, 2002.
- Larman, C. *Agile and Iterative Development: A manager's guide*. Boston, MA, USA: Pearson Education INC., 2004.
- Levine, L. Reflections on Software Agility and Agile Methods: Challenges, Dilemmas, and the Way ahead. *Engineering Institute - Carnegie Mellon University*, 2005.
- Likert, R. A technique for the measurement of attitudes. *Archives of Psychology*, v. 22 140, p. 55, 1932.
- Disponível em <http://psycnet.apa.org/psycinfo/1933-01885-001>

- Maldonado, J. C. *Crterios potenciais usos: Uma contribuio ao teste estrutural de software*. Tese de Doutorado, Universidade de Campinas, 1991.
- Martin, J. *Rapid Application Development*. Macmillan Publishing Co., Inc., 1991.
- McCall, J. A. *An Introduction to Software Quality Metrics*. Petrocelli, 1979.
- Miller, D. C. *Handbook of research design and social measurement*. SAGE Publications Inc., 1991.
- Mnkandla, E.; Dwolatzky, B. Defining Agile Software Quality Assurance. In: *International Conference on Software Engineering Advances (ICSEA'06)*, 2006a.
- Mnkandla, E.; Dwolatzky, B. Defining agile software quality assurance. In: *International Conference on Software Engineering Advances (ICSEA'06)*, 2006b.
- Myers, G. J.; Sandler, C.; Badgett, T.; Thomas, T. M. *The art of software testing*. New Jersey: John Wiley and Sons, 2004.
- Perry, W. *Effective methods for EDI quality assurance*. Prentice Hall, 1987.
- Pfleeger, S. L. *Software Engineering: Theory and Practice*, v. 39. Prentice Hall, 792 p., 2001.
- Pikkarainen, M.; Haikara, J.; Salo, O.; Abrahamsson, P.; Still, J. The impact of agile practices on communication in software development. *Kluwer Academic Publishers*, v. 13, n. 3, p. 303–337, 2008.
- Pressman, R. S. *Software Engineering - A Practitioner's Approach*. 5 ed. McGraw-Hill, 2010.
- Punter, T.; Ciolkowski, M.; Freimut, B.; John, I. Conducting on-line surveys in software engineering. In: *International Symposium on Empirical Software Engineering (ISESE'03)*, IEEE Computer Society, 2003.
- Ramesh, B.; Cao, L.; Mohan, K.; Xu, P. Can distributed software development be agile? *Commun. ACM*, v. 49, n. 10, p. 41–46, 2006.
Disponvel em <http://doi.acm.org/10.1145/1164394.1164418>
- Rapps, S.; Weyuker, E. J. Data flow analysis techniques for test data selection. In: *6th International Conference on Software Engineering*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1982, p. 272–278.

- Rocha, A. R. C. *Um modelo para avaliação da qualidade de especificações*. Tese de Doutorado, PUC Rio, 1983.
- Rocha, A. R. C.; Maldonado, J. C.; Weber, K. C. *Qualidade de Software: Teoria e Prática*. São Paulo: Prentice Hall, 2001.
- Runeson, P.; Höst, M. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, v. 14, n. 2, p. 131–164, 2008. Disponível em <http://www.springerlink.com/index/10.1007/s10664-008-9102-8>
- Schwaber, K.; Beedle, M. *Agile Software Development with Scrum*. Prentice Hall, 2001.
- Sommerville, I. *Software Engineering*. 7th ed. Pearson Addison Wesley, 2004.
- Stake, R. E. *The art of case study research*. Sage Publications, 175 p., 1995. Disponível em <http://books.google.com/books?hl=en&lr=&id=ApGdBx76b9kC&pgis=1>
- Teles, V. M. a. *Um estudo de caso da adoção de práticas e valores do Extreme Programming*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, 2005.
- Travassos, G.; Gurov, D.; Amaral, E. Introdução à Engenharia de Software Experimental. *Relatório Técnico ES59002Abril Programa de Engenharia de Sistemas e Computação COPPEUFRJ*, p. 52, 2002. Disponível em <http://www2.ufpa.br/cdesouza/teaching/topes/4-ES-Experimental.pdf>
- Tripp, D. Pesquisa-ação: uma introdução metodológica. *Educação e pesquisa*, v. 31, n. 3, p. 443–466, 2005.
- Wainer, J. Métodos de pesquisa quantitativa e qualitativa para a ciência da computação. *Kowaltowski TB Karin editor Atualizações em informática*, p. 1–42, 2007. Disponível em <http://www.pucrs.br/famat/viali/educem/material/textos/Pesquisa.pdf>
- Yin, R. K. *Case Study Research: Design and Methods*. 2nd ed. SAGE Publications Inc., 1994.
- Zhu, H.; Hall, P. A. V.; May, J. H. R. Software unit test coverage and adequacy. *ACM Comput. Surv.*, v. 29, n. 4, p. 366–427, 1997.

Questões - Estudo de Caso

A.1 Questões sobre o método de desenvolvimento

As questões sobre o método de desenvolvimento podem variar de acordo com o método utilizado pela empresa. Isso ajuda a validar se a empresa realmente utiliza um processo de desenvolvimento que pode ser relacionado com algum método descrito na literatura. Além disso, é possível adicionar questões baseado no método empregado, por exemplo, se uma empresa utiliza o método ágil XP, deveriam existir atividades de TDD, que podem ser aprofundadas neste tópico.

Desse modo, as seguintes questões foram elaboradas:

1. Qual o método de desenvolvimento utilizado pela empresa ?
2. Quanto tempo a empresa utiliza esse método de desenvolvimento ?
3. A empresa já utilizou ou utiliza algum outro método (ágil ou não) ?
4. Você pode descrever o método de desenvolvimento do início ao fim ?

O propósito desta questão era deixar o entrevistado discorrer sobre o processo de desenvolvimento. Os pontos relevantes desta pergunta eram:

- Identificar estrutura organizacional da equipe de desenvolvimento.

- Verificar se a equipe possui um arquiteto de software, um equipe dedicada de testes e/ou uma equipe dedicada a qualidade de software.
- Descobrir se é gerada alguma documentação sobre o desenvolvimento.
- Ferramentas utilizadas para gestão, testes e controle de qualidade.

A.2 Questões sobre qualidade de software

As questões sobre qualidade de software tiveram foco na qualidade dos produtos gerados pelo processo de desenvolvimento.

5. **Quais métricas de qualidade de software são utilizadas na empresa ? (seja para estimativa de tempo, tamanho de software, gestão de projetos, qualidade, outros)**

A questão é aberta, devido a natureza das operações da empresa, mas esperava-se que pudessem classificar as métricas nos seguintes conjuntos:

- Funcionalidade.
- Confiabilidade.
- Usabilidade.
- Efetividade.
- Manutenibilidade.
- Portabilidade.

6. **Como essas métricas são coletadas e analisadas ?**

7. **A empresa possui atividades específicas para controle ou garantia de qualidade ?**

8. **Como é o processo de testes ?**

Os pontos de interesse desta questão eram:

- Se existe equipe dedicada para testes.
- Quais ferramentas de teste de software são utilizadas.
- Quais critérios de teste são utilizados.
- Como é o processo de aceitação do usuário/cliente.

- Se é utilizado algum servidor de integração contínua. - Um Servidor de Integração Contínua (SIC) é um servidor que executa testes automatizados a cada vez que um código é atualizado no próprio servidor. Ele garante não só a execução constante dos casos de testes criados mas também que novos códigos não gerem erros ao código previamente desenvolvido. Além de relatórios sobre a execução dos testes grande parte dos SIC avisam imediatamente os desenvolvedores se alguma atualização causou erros a execução do código.

Foi dado destaque na atividade de teste pois no planejamento e piloto do estudo de casos percebeu-se grande foco nas atividades de teste e validação de software no desenvolvimento com métodos ágeis. Por esse motivo, o tema foi aprofundado com perguntas específicas e um debate mais amplo sobre seu impacto na qualidade de software.

9. Discorra sobre qualidade de software dentro da empresa.

Como definido anteriormente qualidade de software pode ter significados diferentes para pessoas diferentes. O propósito dessa pergunta é identificar pontos como:

- Percepção do entrevistado sobre qualidade dos produtos da empresa.
- Identificação de ocorrências onde o controle de qualidade ajudou efetivamente o desenvolvimento.
- Identificação de ocorrências onde o teste de software ajudou efetivamente o desenvolvimento.
- Percepção do entrevistado sobre garantia de qualidade, identificando se padrões e regras de desenvolvimento afetam efetivamente o produto final.
- Percepção do entrevistado sobre a visão de clientes e *stakeholders* sobre os produtos da empresa.