Test oracles for systems with complex outputs: the case of TTS systems

Rafael Alves Paes de Oliveira

Rafael Alves Paes de Oliveira

Test oracles for systems with complex outputs: the case of TTS systems

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Márcio Eduardo Delamaro

Co-advisor: Prof^a. Dr^a. Fátima de Lourdes dos Santos Nunes

USP – São Carlos

January 2017

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

0634t Oliveira, Rafael Alves Paes de
Test oracles for systems with complex outputs:
the case of TTS systems / Rafael Alves Paes
de Oliveira; orientador Márcio Eduardo Delamaro;
corientadora Fátima de Lourdes dos Santos Nunes. -
São Carlos - SP, 2015.
212 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2015.

1. Test oracle. 2. Software testing. 3. Test
automation. 4. Systems with complex outputs.
5. Text-to-speech systems. I. Delamaro, Márcio
Eduardo, orient. II. Nunes, Fátima de Lourdes dos
Santos, coorient. III. Título.

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Rafael Alves Paes de Oliveira

Oráculos de teste para sistemas com saídas complexas: o caso dos sistemas TTS

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor Program in Computer Science and Computational Mathematics. *EXEMPLAR DE DEFESA*

Área de Concentração: Computer Science and Computational Mathematics

Orientador: Prof. Dr. Márcio Eduardo Delamaro

Corientadora: Prof^a. Dr^a. Fátima de Lourdes dos Santos Nunes

USP – São Carlos

To my lovely wife Juliana and our happy son Nicholas

ACKNOWLEDGEMENTS

First of all, I would like to thank my beautiful wife Juliana for being an essential part of this research, tolerating my absences and being, at times, mom and dad of our son. Lucky me to find you. A special thank my smiling son Nicholas for repeatedly recharging my energies with his sweetness, hugs, and kisses - page after page. You two are the center of my life and my reason to keep going further.

I'm grateful to my family: my strong dad Doni, my sweet mom Gorete, and my shine-confident sister Camila (Biba). Thank you for teaching me repeatedly everything that books and research do not show. I fell thank every day for being raised by you, learning over and over through your examples.

During my undergrad, the destiny was generous and put in my way precious people: my advisor Prof. Dela, and my co-advisor Prof. Fatima. Thanks for your untiring teachings, advice, and suggestions. @ Dela, thanks for being a special friend in your spare times of adviser.

Thanks, Prof. Atif Memon, for your recommendations and your contributions to this dissertation. You made my journey in Maryland at UMD much more valuable.

I'm also thanks for my uncountable friends from Marília, São Carlos, Washington-DC, Dois Vizinhos.

I'd like to thanks my friends from LABES. I'll not mention any of you to avoid committing injustices.

A special thank to all my Professors from ICMC/USP. You are the best ones.

Thanks to all of my colleagues from UTFPR câmpus Dois Vizinhos.

Finally, I'd like to thank some of the reviewers of this manuscript: Norine Walker and Erico Milani.

*“We cannot solve our problems
with the same thinking we used when we created them.”
(Albert Einstein)*

ABSTRACT

OLIVEIRA, R. A. P.. **Test oracles for systems with complex outputs: the case of TTS systems**. 2015. 212 f. Doctoral dissertation (Doctorate Candidate Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Software testing is one of the most important Software Engineering processes, being the primary activity to check the conformance between the software requirements and its actual behavior. The automation of software testing activities is essential to certify productivity and effectiveness in such activities. Test automation leads testing activities to be conducted under systematic and accurate criteria, raising the chance of testers to reveal faults or inconsistencies. Test oracles are elementary members in software testing automation, being the mechanism responsible for indicating the correctness of software outputs. In testing environments, test oracles can be effectively implemented based on several sources of information about the *Software Under Testing* (SUT): software specifications, assertions, formal methods (*Finite State Machines* (FSM)), formal specifications, etc, machine-learning methods, and metamorphic relations. Regardless of the implementation strategy, test oracles are vulnerable to false positive/negative verdicts, configuring what the literature describes as “the oracle problem”. Therefore, test oracles are a non-trivial and challenging object of studies of the software engineering research area. SUT’s outputs in unusual formats make it harder the oracle problem. Audio, images, three-dimensional objects, virtual reality environments, complex statistical compositions, etc, are examples of non-trivial output formats. In the software testing context, SUTs with unusual outputs can be called “complex-output systems”. In this doctorate dissertation, we propose and evaluate a novel test oracle approach for complex-output systems called *feature-based test oracles*. The purpose of feature-based test oracles is the appropriation of a processing image technique called *Content-Based Image Retrieval* (CBIR) to collect information from features extracted from the SUT’s outputs to compose test oracles. Given a query image, CBIR combines feature extraction and similarity functions to alleviate the problem of searching for digital images in large databases. In previous research, we have integrated CBIR concepts in a testing framework to support the automation of testing activities in processing image systems and systems with *Graphical User Interfaces* (GUI). In this doctorate dissertation, we extended that framework and its concepts to general complex-output systems, addressing the feature-based test oracle approach. We use *Text-To-Speech* (TTS) systems to validate empirically our test oracle technique. Through the results of five empirical analyses, three of them conducted in line with problems of a real-world industry TTS system, show the proposed technique is a valuable instrument to automate testing activities and alleviate practitioners’ efforts on testing complex output systems. We conclude the proposed test oracles are effective because they systematically evaluate the SUT’s sensorial output rather than produce verdicts based on subjective specifications. As future work, we plan

to conduct investigations towards the reduction of false positives/negatives and the association of the test oracles with machine learning techniques and metamorphic relations.

Key-words: Test oracle, Software testing, Test automation, Systems with complex outputs, Text-to-speech systems.

RESUMO

OLIVEIRA, R. A. P. **Test oracles for systems with complex outputs: the case of TTS systems.** 2015. 212 f. Doctoral dissertation (Doctorate Candidate Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Teste de Software é um dos processos mais importantes da Engenharia de Software, sendo a principal atividade para averiguar a conformidade de requisitos de software e suas saídas. A automatização das atividades de teste é essencial para conferir produtividade e efetividade em tais atividades. A automatização faz com que atividades de teste sejam conduzidas sob critérios sistemáticos e precisos, aumentando a chance dos testadores de revelarem falhas ou inconcistências. Oráculos de teste são membros elementares na automatização do teste de software, sendo o mecanismo responsável por indicar a corretude das saídas do software. Em ambientes de teste, oráculos de teste podem ser efetivamente implementados com base em diversas fontes de informação sobre o sistema em teste: especificações de software, assertivas, métodos formais (máquinas de estados finitas, especificações formais, etc), métodos de aprendizagem de máquina e relações metamórficas. Independente da estratégia de implementação, oráculos de teste são vulneráveis a veredictos de falsos positivos/negativos, configurando o que é apresentado na literatura como “O problema do Oráculo”. Então, na área de engenharia de software, oráculos de teste são objetos de estudo não-triviais e desafiadores. O problema de oráculo é potencializado quando as saídas do sistema em teste são dadas em formatos não triviais como, por exemplo, áudio, imagens, objetos tridimensionais, ambientes de realidade virtual, composições estatísticas complexas, etc. No contexto do teste de software, sistemas com saídas não triviais podem ser chamados de “sistemas com saídas complexas”. Esta tese de doutorado propõe e avalia uma nova estratégia de oráculo de teste para sistemas com saídas complexas. O propósito de tal estratégia é a apropriação da técnica de processamento de imagem conhecida como CBIR (*Recuperação de Imagem Baseada em Conteúdo* – CBIR) para coletar informações de características extratidas do sistema em teste, compondo oráculos de teste. A partir de uma imagem de busca, o CBIR combina extração de características e funções de similaridade para aliviar problemas de busca em grandes bases de imagens digitais. Em pesquisas anteriores, conceitos de CBIR foram integrados em um arcabouço de teste para apoiar a automatização de atividades de teste em sistemas de processamento de imagens e sistemas com interfaces gráficas. Esta tese de doutorado estende o arcabouço e seus conceitos para sistemas com saídas complexas em geral. Sistemas Texto-Fala (TTS) foram utilizados para validações empíricas. Os resultados de seis análises empíricas, duas delas condizidas em consonância com problemas de um TTS industrial, revelam que a técnica proposta é um valioso instrumento para automatizar atividades de teste e aliviar esforços de profissionais da indústria ao teste sistemas com saídas complexas. Conclui-se que a efetividade dos oráculos de teste propostos são devido às sistemáticas análises

do conteúdo das saídas dos sistemas em teste, em vez da análises de especificações subjetivas. Os trabalhos futuros vislumbrados devem ser conduzidos no intuito de reduzir número de falsos positivos/negativos e a associação dos oráculos de teste com técnicas de aprendizado de máquina e relações metamórficas.

Palavras-chave: Oráculo de teste, Teste de software, Automatização de teste, Sistemas com saídas complexas, Sistemas Texto-Fala.

LIST OF FIGURES

Figure 1 – A Concept Map of Software Testing	43
Figure 2 – A Generic Model of a Black-box Testing Process.	44
Figure 3 – Generic Test Oracle Structure	45
Figure 4 – Test Oracles Using Different Sources of Information	45
Figure 5 – State-Machines as Test Oracles	58
Figure 6 – Analysis of Published Research on Test Oracles.	63
Figure 7 – Pie Chart of Test Oracle Approaches	64
Figure 8 – Year by Year Evolution of Oracle Approaches.	65
Figure 9 – Comparative of Oracle Approaches Year by Year.	66
Figure 10 – TTS’s generic structure	72
Figure 11 – Generic structure of TTS systems	73
Figure 12 – Speech signal of the same word with different accents in the frequency-domain	75
Figure 13 – Signal of the same sentence with different intonation	76
Figure 14 – Search string of the SM	78
Figure 15 – <i>Workflow</i> followed in the mapping study	79
Figure 16 – Studies selection for each database	80
Figure 17 – Methods to evaluate the quality of synthesized speech	82
Figure 18 – Studies categorization by research nature and contributions	83
Figure 19 – Challenges revealed by the SM	84
Figure 20 – Future work trend revealed by the SM	85
Figure 21 – Studies selection by venue	86
Figure 22 – Studies selected by year of publication	86
Figure 23 – Quantitative information about the SM	87
Figure 24 – O-FIM/CO General Structure.	95
Figure 25 – Generic Instance of an Oracle Description.	96
Figure 26 – Generic workflow associated with the usage of O-FIm/CO	99
Figure 27 – Audio signal in the frequency domain and its formants	109
Figure 28 – Chart of Brazilian Portuguese vowels	110
Figure 29 – Overall analysis of the test oracles	122
Figure 30 – Normal distributions of the feature-based test oracle classifications	150

LIST OF ALGORITHMS

Algoritmo 1 – Test Oracle Using an “xUnit” Framework.	46
Algoritmo 2 – Excerpt from an Assertion Code.	55
Algoritmo 3 – A Program to be Tested Using Interface Contracts	55
Algoritmo 4 – Code Representing Class to be Tested	56
Algoritmo 5 – A Sample of SUT’s output Regarding State Machines as Oracle.	57
Algoritmo 6 – Abstract class for O-FIm/CO plug-ins	100
Algoritmo 7 – Java interface for O-FIm/CO similarity function	101
Algoritmo 8 – Source code of an audio feature extractor for O-FIm/CO	102
Algoritmo 9 – Spectral centroid extractor	103
Algoritmo 10 – Compactness extractor	104
Algoritmo 11 – Spectral flux	104
Algoritmo 12 – Zero Crossing extractor	105
Algoritmo 13 – Traditional Area Moment extractor	106
Algoritmo 14 – Main method of the Vowel extractor	111
Algoritmo 15 – Main method of the Phoneme extractor	112

LIST OF TABLES

Table 1 – Implications for Manual and Automated Testing.	39
Table 2 – Comparison Between Manual and Automated Testing.	40
Table 3 – Testing tools, frameworks for testing and their application	41
Table 4 – Compilation of Testing Oracles by Their Automation Process.	52
Table 5 – Number of SUTs and Categories.	66
Table 6 – Methods to be implemented by an O-FIm/CO plug-in	97
Table 7 – Brazilian Portuguese Vowels and respective examples	109
Table 8 – Metrics collect from the project of the extractors	111
Table 9 – Template of answers collected from the pre-defined dataset	113
Table 10 – Summary for the Vowel Feature Extractor.	114
Table 11 – Summary for the Phoneme Feature Extractor.	114
Table 12 – Binomial test performed at a 95% confidence interval	115
Table 13 – Subject TTS used in the study	119
Table 14 – Data collected using Female-German voices in Google TTS	121
Table 15 – Overall results considering all samples	121
Table 16 – Precision/Recall by language	123
Table 17 – Sentences synthesized and their correspondent hand-normalized versions	127
Table 18 – TTS subjects explored to check test oracles against normalization issues	130
Table 19 – Detailed results of Oracle 2 applied in CPqD TTS	131
Table 20 – Overall results for CPqD TTS considering both oracles	132
Table 21 – Overall results for Ivona TTS considering both oracles	133
Table 22 – Overall results for ODCAST TTS considering both oracles	134
Table 23 – Overall results for VOCALWARE TTS considering both oracles	135
Table 24 – General results and metrics collected from the study	136
Table 25 – Overall results for VozMe TTS considering both oracles	137
Table 26 – Template with expected verdicts for automated test oracles	143
Table 27 – My caption	143
Table 28 – Statistical analysis of feature-based test oracles to evaluate TTSs’ outputs with small audio databases	148
Table 29 – Template with expected verdicts of prosody study	155
Table 30 – Overall results of feature-based test oracle evaluating prosody problems	156
Table 31 – Binomial statistical test of feature-based test oracles evaluating prosody problems	162
Table 32 – Oracle 01 – CPqD – Feature-based test oracles against normalization issues	196

Table 33 – Oracle 02 – CPqD – Feature-based test oracles against normalization issues .	197
Table 34 – Oracle 01 – Ivona – Feature-based test oracles against normalization issues .	198
Table 35 – Oracle 02 – Ivona – Feature-based test oracles against normalization issues .	199
Table 36 – Oracle 01 – ODDCAST – Feature-based test oracles against normalization issues	200
Table 37 – Oracle 02 – ODDCAST – Feature-based test oracles against normalization issues	201
Table 38 – Oracle 01 – VOCALWARE – Feature-based test oracles against normalization issues	202
Table 39 – Oracle 02 – VOCALWARE – Feature-based test oracles against normalization issues	203
Table 40 – Oracle 01 – VozMe – Feature-based test oracles against normalization issues	204
Table 41 – Oracle 02 – VozMe – Feature-based test oracles against normalization issues	205
Table 42 – Oracle 01 – Feature-based test oracles evaluating TTT with reduced audio database	208
Table 43 – Oracle 02 – Feature-based test oracles evaluating TTT with reduced audio database	209
Table 44 – Oracle 03 – Feature-based test oracles evaluating TTT with reduced audio database	210
Table 45 – Oracle 04 – Feature-based test oracles evaluating TTT with reduced audio database	211
Table 46 – Oracle 05 – Feature-based test oracles evaluating TTT with reduced audio database	212

LIST OF ABBREVIATIONS AND ACRONYMS

ACM		<i>Association for Computing Machinery</i>
ADL		<i>Architecture Description Language</i>
AI		<i>Artificial Intelligence</i>
ANN		<i>Artificial Neural Networks</i>
AO		<i>Aspect-Oriented</i>
API		<i>Application Program Interface</i>
ASCII		<i>American Standard Code for Information Interchange</i>
CAD		<i>Computer-aided diagnosis</i>
CBAR		<i>Content-Based Audio Retrieval</i>
CBIR		<i>Content-Based Image Retrieval</i>
CC		<i>McCabe Cyclomatic Complexity</i>
CoMIRVA		<i>Collection of Music Information Retrieval and Visualization Applications</i>
CPqD		<i>Centro de Pesquisa e Desenvolvimento em Telecomunicações</i>
DRT		<i>Diagnostic Rhyme Test</i>
FFT		<i>Fast Fourier Transform</i>
FSM		<i>Finite State Machines</i>
GMM		<i>Gaussian Mixture Models</i>
GUI		<i>Graphical User Interfaces</i>
HCI		<i>Human-Computer Interaction</i>
HMM		<i>Hidden Markov Model</i>
HMM		<i>Hidden Markov Model</i>
IEEE		<i>Institute of Electrical and Electronics Engineers</i>
IP		<i>Image Processing</i>
IPO		<i>input–process–output</i>
ITU		<i>International Telegraph Union</i>
JMF		<i>Java Media Framework</i>
JML		<i>Java Modeling Language</i>
JVM		<i>Java Virtual Machine</i>
L&F		<i>Look and Feel</i>
LOC		<i>Lines Of Code</i>
LPC		<i>Linear Prediction Coefficients</i>

M-mp *M-model program testing*
MB *MegaBites*
MFCC ... *Mel-Frequency Cepstral Coefficients*
ML *Machine Learning*
MOS *Mean Opinion Score*
MP3 *MPEG-1/2 Audio Layer 3*
MR *Metamorphic Relations*
MRT *Modified Rhyme Test*
O-FIm/CO *Oracle For Images and Complex Outputs*
OCL *Object Constraint Language*
OMG *Object Management Group*
OO *Object-Oriented*
QR *Quick Response*
RQ *Research Questions*
RQ *Research Question*
SBC *Sociedade Brasileira de Computação*
SE *Software Engineering*
SLA *Supervised Learning Machines*
SLR *Systematic Literature Reviews*
SM *Systematic Mapping*
SUD *Software Under Development*
SUT *Software Under Testing*
TDA *Traditional Area Method of Moments*
TTS *Text-To-Speech*
UML *Unified Modeling Language*
V&V *Validation and Verification*
VR *Virtual Reality*
WAVE *WAVEform Audio File Format*

CONTENTS

1	INTRODUCTION	27
1.1	Motivation	28
1.2	Methodology and Objectives	30
1.3	Research Scope, Contributions, and Limitations	31
1.4	Dissertation outline	33
2	BACKGROUND	35
2.1	Overview	35
2.2	Software testing	35
2.2.1	<i>Concepts and theories</i>	36
2.2.2	<i>Automated testing</i>	38
2.2.3	<i>Supporting tools</i>	40
2.3	A holistic view of test oracles	41
2.3.1	<i>Definitions and concepts</i>	42
2.3.2	<i>The oracle problem</i>	46
2.3.3	<i>Test oracle trade-offs</i>	47
2.3.4	<i>Taxonomies</i>	48
2.3.4.1	<i>Generic classification</i>	49
2.3.4.2	<i>Specific Taxonomies</i>	50
2.3.4.3	<i>A Taxonomy by Oracle Information Characteristics (OLIVEIRA; KANEWALA; NARDI, 2014)</i>	53
2.3.5	<i>Mapping study and historical evolution</i>	62
2.3.5.1	<i>Study selection strategy</i>	62
2.3.5.2	<i>Study classification</i>	63
2.3.5.3	<i>Result analysis</i>	63
2.4	Complex-output systems	66
2.4.1	<i>Testing challenges</i>	68
2.5	Final remarks and overview	69
3	TTS - A COMPLEX-OUTPUT SYSTEM	71
3.1	Overview	71
3.2	TTS: a complex-output system	71
3.2.1	<i>Concepts and definitions</i>	71

3.2.1.1	<i>Speech signal processing</i>	74
3.2.2	<i>Quality limitation</i>	75
3.3	Testing strategies and quality evaluation methods for TTS systems: a mapping study	77
3.3.1	<i>Motivation</i>	77
3.3.2	<i>Research Protocol</i>	78
3.3.3	<i>Systematic Mapping Results</i>	79
3.3.4	<i>Statistical data and demographic information</i>	85
3.3.5	<i>General findings</i>	86
3.4	Final remarks and overview	88
4	THE FEATURE-BASED TEST ORACLE APPROACH	89
4.1	Overview	89
4.2	Feature-based test oracles – designing automated test oracle to complex-output systems	89
4.2.1	<i>CBIR – Content-Based Image Retrieval</i>	90
4.2.2	<i>Requirements and Definitions</i>	91
4.2.3	<i>Classification and particularities of feature-based test oracles</i>	92
4.2.4	<i>When and how to use feature-based test oracles?</i>	93
4.3	The O-Flm/CO framework	94
4.3.1	<i>O-Flm/CO's structure</i>	95
4.3.2	<i>O-Flm/CO's API</i>	96
4.3.3	<i>Previous applications and O-Flm/CO in Practice</i>	97
4.4	O-Flm/CO to TTS systems	99
4.5	Final remarks and overview	106
5	EMPIRICAL EVALUATION	107
5.1	A pilot study on test oracles for TTS systems using feature extrac- tors for formants and phonemes	107
5.1.1	<i>Feature extractors implemented</i>	108
5.1.1.1	<i>Vowel extractor</i>	108
5.1.1.2	<i>Phoneme extractor</i>	110
5.1.2	<i>Experimental Setup and Research Question</i>	112
5.1.3	<i>Experimental Results</i>	114
5.1.4	<i>Experimental Analysis</i>	115
5.1.5	<i>Threats to Validity</i>	116
5.1.6	<i>Recommendations and Findings</i>	116
5.2	Towards on defining effective test oracles to TTS systems	117
5.2.1	<i>Research Questions</i>	117
5.2.2	<i>Experimental Design</i>	117

5.2.3	<i>Subject TTSs</i>	119
5.2.4	<i>Experimental procedure</i>	119
5.2.5	<i>Experiment conduction</i>	120
5.2.6	<i>Experimental Results and Discussion</i>	120
5.2.7	<i>Threats to validity</i>	123
5.2.8	<i>Final remarks</i>	124
5.3	<i>Automated text oracles to check normalization problems in TTS systems</i>	124
5.3.1	<i>Research Questions and Metrics</i>	126
5.3.2	<i>Experimental Design and subjects</i>	126
5.3.3	<i>Subjects</i>	130
5.3.4	<i>Experimental Results and Discussion</i>	130
5.3.5	<i>Threats to Validity</i>	136
5.3.6	<i>Final Remarks and Findings</i>	138
5.4	<i>Feature-based test oracles to evaluate synthesized speech from TTSs with reduced audio databases</i>	138
5.4.1	<i>Research Questions</i>	140
5.4.2	<i>TTS Subject</i>	140
5.4.3	<i>Experimental Design and Procedure</i>	140
5.4.4	<i>Results and Discussion</i>	143
5.4.5	<i>Threats to Validity</i>	151
5.4.6	<i>Findings and Recommendations</i>	152
5.5	<i>On evaluating feature-based test oracles to check artificial prosody problems of TTS systems</i>	152
5.5.1	<i>Research Questions and Metrics</i>	153
5.5.2	<i>Subjects</i>	154
5.5.3	<i>Experimental Design and Procedure</i>	154
5.5.4	<i>Experimental Results and Discussion</i>	156
5.5.5	<i>Threats to Validity</i>	163
5.5.6	<i>Final Remarks and overall recommendation</i>	164
5.6	<i>Final remarks and overview</i>	165
6	CONCLUSIONS AND FUTURE DIRECTIONS	167
6.1	<i>Discussion</i>	168
6.1.1	<i>Necessity of specific domain knowledge</i>	168
6.1.2	<i>Feature extractors dependency</i>	168
6.1.3	<i>Dependency of an output reference</i>	169
6.1.4	<i>Amortization of costs</i>	169
6.1.5	<i>Feature-based test oracles' limitation</i>	170
6.2	<i>Summary of contributions</i>	171

6.3	Future work	172
6.4	Final remarks	172

BIBLIOGRAPHY	175
------------------------	-----

APPENDIX A	RESULTS OF FEATURE-BASED TEST ORACLES EVALUATING NORMALIZATION ISSUES IN TTS SYSTEMS (SEC. 5.3	195
------------	--	-----

APPENDIX B	RESULTS OF FEATURE-BASED TEST ORACLES EVALUATING TTS WITH REDUCED AUDIO DATABASES (SEC. 5.4)	207
------------	--	-----

INTRODUCTION

In the last decade, the *Software Engineering* (SE) area has evolved tremendously. This evolution follows the evolution of the computation systems and has increased the intrinsic competitiveness in the software industry (EBERT; KUHRMANN; PRIKLADNICKI, 2016). Researchers and practitioners have dedicated efforts on reducing the software development life cycle, supplying the need for contemporary design processes, defining faster-inexpensive testing strategies, and proposing effective maintenance strategies (SäGESSER; JOSEPH; GRAU, 2013). In addition, research in the SE area is ordinarily conducted under three aspects: (i) theoretical aspects, (ii) experimental aspects, and (iii) aspects of automation.

Software testing is one of the most important components of the SE area, contributing to the *Validation and Verification* (V&V) phase of software development (BERTOLINO, 2007a). Testing activities are key elements to ensure the reliability and quality of software products, making it more trustworthy. Conceptually, software testing is the dynamic activity of running (SUT) (*Software Under Testing*) using specified inputs which decide whether the outputs are correct or not (AMMANN; OFFUTT, 2008a). Testing techniques and criteria support testers on how to design effective testing routines (DELAMARO; MALDONADO; JINO, 2007). The automation of testing activities can potentially reduce costs and efforts during the development process. Contemporary testing activities have been evolving significantly to follow the evolution of software products and the SE processes (HARMAN; JIA; ZHANG, 2015; MUCCINI; FRANCESCO; ESPOSITO, 2012). This evolution has been increasing the challenges of designing effective testing activities on several contemporary systems.

In the context of software testing, the evolution of technologies leads the current applications to produce complex outputs. Complex-output systems are those whose outputs severely limit the application of automated testing techniques and strategies. Some examples of contemporary complex-output systems are: (1) systems with complex GUIs whose visual information is reflected in different screen resolution, sizes, and orientations; (2) TTS systems that convert written text in audio files; (3) some Web application quality depends on the browsers; (4) Mobile

apps that run over several different platforms under different screen resolutions, sizes, and orientations of multiple devices; and (5) *Virtual Reality* (VR) environments, and others. Five main domains can represent the output of a complex-output system: (i) synthetic audio; (ii) processed/rendered images; (iii) three-dimensional objects; (iv) virtual reality environments; and (v) complex statistical numbers. Complex-output systems are becoming standard and the software industry is developing more attractive and exquisite applications.

This dissertation aims provide, evaluate, and validate a novel testing support strategy to contemporary complex-output systems through automated test oracles, fomenting new testing research for these domains. In automated software testing environments, test oracles represent the mechanism to evaluate whether an SUT execution is correct or not (HOFFMAN, 2006; HARMAN *et al.*, 2013; BARR *et al.*, 2015). Two main reasons drive this study: (i) the mainstream adoption of complex-output systems is severely limited by their quality; and (ii) new investigations on systematic testing strategies to complex-output systems may instigate the research community to derive testing solutions to contemporary systems with similar outputs domains.

Due to the complexity of tasks on automating effective testing strategies to complex-output systems and due to the variety of complex-output domains, our dissertations' goals are broad and they cannot be settled in the standard fashion. For these reasons, this research focuses on producing contributions towards approaching SE activities, in particular, software testing, to complex-output systems. Our research supplies four main contributions: (i) the proposal of a test oracle approach called feature-based. This approach addresses to implementing testing activities in complex-output systems; (ii) the extension and generalization of a practical framework to meet feature-based approach; (iii) a set of empirical validations on composing flexible automated test oracles for complex-output systems; and (iv) a massive discussion on test automation for contemporary uncommon domains.

1.1 Motivation

Two main reasons drive this dissertation: (i) the mainstream adoption of complex-output systems is severely limited by their quality; and (ii) new investigations on systematic testing strategies to complex-output systems may instigate the research community to derive testing solutions to contemporary systems with similar outputs domains.

Increasing the number of tasks that are feasible to be performed by the current computational systems has led several computational systems to become an indispensable technology for science, business, engineering, and medical applications (SOMMERVILLE, 2010). The dependency of computational systems to perform several regular activities increases the importance of the system's reliability. In the current society, a system's reliability is an elementary requirement for the extensive adoption of the software systems (DURELLI *et al.*, 2012). Hence,

V&V activities must be applied in several points of the *Software Under Development* (SUD)’s life cycle (MYERS; SANDLER; BADGETT, 2011).

However, V&V activities are often a time consuming and costly phase in software development processes (REPASI, 2009). Besides the lack of proper tools, establishing adequate and systematic testing strategies is a challenge faced by practitioners. This sort of challenge often leads testers to apply manual, and ad-hoc testing strategies, making V&V activities unproductive, ineffective, and biased (MYERS; SANDLER; BADGETT, 2011). A well-known resource to raise the productivity and quality of the V&V activities is the application of automated testing strategies. Automated testing strategies often involve a threefold process: (i) taking into account the SUT’s specification to select adequate test data (automatically or manually); (ii) setting and automating an environment or tool to run an SUT with the test data; (iii) selecting a method of determining the correctness of the outputs; and (iv) checking the results through report or pre-defined coverage criteria.

A limiting factor of applying automated testing strategies is the complexity of the outputs of some contemporary systems (DELAMARO; MALDONADO; JINO, 2016). In software systems, outputs are data produced by a system processing from a given input (BERTOLINO, 2007a). Thus, outputs are the information produced by computational systems and consumed by users. Systems produce outputs in two different levels: (i) *software level*; and (ii) *physical level*. Software-level outputs could be, for example, a number generated from a given processing. On the other hand, physical-level outputs can be, for example, printed documents or, even, an object produced by three-dimensional printing (3D printing). The fact is that outputs can vary from simple numbers, Strings, and pre-defined objects to complex sounds (audio), processed images, and three-dimensional environments (OLIVEIRA, 2014). Several of the contemporary software are complex-output systems, producing some complex output domain and limiting the application of effective testing strategies.

This dissertation is motivated by a lack of proper automated testing strategies for complex-output systems. It is necessary to approach SE techniques and complex-output systems through solid scientific contributions. Software testing strategies to these domains remain a problem. Researchers and practitioners must define several key factors for testing complex-output systems: (i) define what to test; (ii) define acceptable outputs; (iii) define patterns and behaviors that are acceptable or not; (iv) identify reliable sources of information for testing; and (v) provide proper environments to run the test data.

A study from the Brazilian Computer Society (*Sociedade Brasileira de Computação* (SBC) – from Portuguese) created an extra motivation for this study (SBC, 2009). The aforementioned study presents results from a seminar of discussions about the challenges to be faced by computing in Brazil until 2020. The “development of reliable systems” is one of the challenges widely discussed. This emerging topic is associated with the application of proper V&V activities of the current systems. Then, the SBC’s study motivates and guides this study leading

our research to address a non-trivial topic – to automate testing strategies of complex-output systems.

1.2 Methodology and Objectives

The methodology employed in this research adapts concepts of *Content-Based Image Retrieval* (CBIR) to promote the automation of test oracles for complex-output domains. CBIR is a processing image technique that uses feature extraction and similarity functions to identify and rank similar images from an image database, taking into account a query image (BUENO *et al.*, 2002; GUDIVADA; RAGHAVAN, 1995). In the methodology of our research, CBIR concepts are implemented through a framework called *Oracle For Images and Complex Outputs* (O-Flm/CO). O-Flm/CO is a software framework that provides an environment capable of performing feature extraction and objects comparison as automated test oracles.

O-Flm/CO technically allows testers to install plug-ins to act as feature extractors and similarity functions collecting information from complex-output systems in a software testing environment. Through an *Application Program Interface* (API), O-Flm/CO provides a Java program to systematically compare a series of complex outputs, responding whether they are similar in accordance to a specification and a threshold. In our previous work, O-Flm/CO originally addressed to support automated test oracles for graphical outputs (OLIVEIRA; DELAMARO; NUNES, 2010), medical solutions (BERGAMASCO *et al.*, 2015), and web applications (DELA-MARO; NUNES; OLIVEIRA, 2013). Further, in the context of this dissertation, we extended O-Flm/CO to support testing automation of other complex-output domains.

A long-term objective of this study is to improve the quality evaluation of complex-output systems, providing scientific evidence to further studies on software testing for specific complex-output domains. Thus, this dissertation aims at complementing and supporting the state of the art of the SE area, generalizing the concepts of a framework to automate test oracles for systems with specific complex-output domains. Our dissertation's objective is guided by three main factors: (i) complex-output systems are becoming pervasive and common day-after-day; (ii) a lack of studies and tools that effectively approaches V&V activities and complex-output systems; and (iii) the recurrent need for automated SE methods for novel applications.

Our short-term objective is to evaluate the effectiveness of using features from complex outputs to compose automated test oracles. Then, the following specific objectives are defined:

- to define how to use feature extraction to provide information about the correctness of TTS systems through the feature-based test oracle approach;
- to document and generalize features and particularities of TTS systems that can be used in software testing activities; and

- to generalize a framework to test complex-output systems.

1.3 Research Scope, Contributions, and Limitations

There are several complex-output domains and their evaluation in a single study is infeasible. The scope of this dissertation, in terms of its validation, is a particular complex-output domain: *Text-To-Speech* systems (TTS). TTS systems compose software applications that convert written text in audio outputs that represents a human reading that text (TAYLOR, 2009). TTS technologies are, typically, implemented through concatenative (GOPI *et al.*, 2013; LEE; JUNG; KANG, 2007) or parametric strategies (TAKAMICHI *et al.*, 2016; QIAN *et al.*, 2014). Concatenative TTS implementations, after some text normalization strategy, integrate several pieces of recorded audio in a final audio output (TAYLOR, 2009). Parametric TTSs uses a *Hidden Markov Model* (HMM) that creates a sequence of discrete time observations through refined and specific algorithms (WAN *et al.*, 2014).

TTS systems are useful in a plenty of activities to support *Human-Computer Interaction* (HCI) in several applications: auto-organized call centers (CONTOLINI; STOLMENOV; JUNQUA, 2004), e-book reading (ANGUERA *et al.*, 2011), helping blind people to interact better with computers (PORTELE; KRAMER, 1996; SALEOUS *et al.*, 2016; DOBRISEK *et al.*, 1999; SAMMOUDA; ALRJOUNB, 2015), automatic reading news systems for embedded systems (KARABETSOS *et al.*, 2009; ALROUQI *et al.*, 2015), trip orientation and guidance in travel applications (K; P, 2015), etc. Despite the applicability of TTSs in several contemporary occasions, the wide adoption of TTS applications is severely limited due to their low quality and due to the lack of portability associated with high-quality TTS applications. Some of the TTS challenges treated in this dissertation are:

- *An automated alternative to check normalization problems in TTS applications*: a common problem of TTS applications is the text normalization that should be done before the voice synthesizer. Generally, TTS schemes have a set of text-to-text rules to convert acronyms, numbers, fractions, dates, and others, to text format. This step is one of the most problematic aspects of TTS systems and they are manually checked by developers/practitioners through subjective listening tests. This study provides test oracles that are able to automate this process, alleviating human efforts;
- *Trade-off analysis on the size of the TTSs' database*: portability and ubiquity issues are a well-know challenge to TTS applications. Large databases generally have acceptable quality, however their size are not suitable to all of the devices and mobile applications. Small databases do not present quality enough to be wide adopted by different applications. Our approach presented in this study defines automated test oracles that promote a quality evaluation strategy to indicates the quality difference among long and short databases;

- *Source of information for automated quality evaluation of TTS applications:* features extracted from the SUT's output can set a valuable source of information for testing and verification activities. Thus, this study contribute with a novel testing paradigm since it show how to use explore features from the SUT's outputs to automate testing activities;
- *The need for automated SE methods during the developing process of TTS systems:* regarding automated software engineering processes, this dissertation presents an alternative to manual and ad-hoc quality evaluation methods for TTS applications, providing alternatives to automatically capture input and evaluate outputs; and
- *Defining features to be used in testing activities:* the feature extractors and similarity functions provided in this study can be adapted to be useful source of information for testing and verification activities in other contexts.

The main contributions of this dissertation are: (i) our study approaches SE methods in particular, testing activities, and complex-output domains providing an effective testing strategy; (ii) this dissertation provides a broad research on how to deal with complex-output systems regarding automated testing activities; (iii) our study extends a flexible testing framework from a specific-output domain to a broad application-domain in the context of complex-output systems; (iv) this study provides a catalog of feature extractors for synthesized audio that can be adapted to several testing activities, including their source code; (v) this study presents a feasible strategy to explore multidisciplinary resources to alleviate specific-domains problems, adjusting a strategy that can be extended and employed in other problems; (vi) this study provides a broad study on test oracles regarding their taxonomy and trends; (vii) starting from TTS applications, we provided an extension of concepts and guidelines that can be followed to implement automated testing activities to complex-output domains; (viii) trough our study we provided a bird-eye view of possible solutions to automate most of the SE activities to complex-output domains; and (ix) the experiences and the discussion provided in this study allows testers to think outside the box and see novel systematic testing strategies.

The research conducted through this doctorate dissertation has two types of positive implications to software developers:

- (i) short-term implications: at first, the mainstream adoption of the testing strategies provided by this study are associated with the amortization of testing costs. Hence, it is necessary to implement specific-domain feature extractors, adjust an automated testing environment, collect reliable source of information to extract features, define automated strategies to collect actual outputs, and defines rules, descriptions and references for test oracles. In this sense, much of the activities consist of hands-on work in order to provide a proper environment for testing activities. It is possible that the testers need some time to solve technical problems or even other issues that are unpredictable; and

- (ii) long-term implications: regarding the results that might be reach in the future from the wide adoption of this technique, one can mention a favorable testing environment to perform automatic and systematic testing activities several times. In addition to that, tester can improve their testing strategies through the definition of new flexible test oracles in other to automatically validate some important aspect of the application.

On the other hand, as usual, some difficulties are imposed to its adoption: (i) the implementation of specific-domain feature extractors may need support of domain specialists; (ii) the oracle problem still remains. Overall, testers must to deal with “false positives” and “false negatives”. In this sense, depending on the output domain, specific studies can be conducted in order to mitigate false positives/negatives; (iii) another limitation of our study is the need for reliable and trustworthy outputs from which test oracles can extract features to define their verdicts. This is a limitation factor because several complex-output domains do not have any reference of correct output, for instance, biomedical for genetic applications; and (iv) the application of the technique in some domains are unpredictable and, depending on the context, the trade-off of applying our strategy can demand to much effort, making it costly. Fox example, when there are no feature extractors already implemented to given domain.

1.4 Dissertation outline

Besides this introductory chapter, the other content of this study are:

Chapter 2: presents an overview about testing activities and test oracles. Further, concepts of complex-output systems are presented;

Chapter 3: present TTS application regarding their testing challenges and characterization it as complex-output domain;

Chapter 4: presents the feature-based test oracle approach and the extension of the testing framewrok O-FIm/CO to address to feature-based concepts;

Chapter 5: five empirical evaluations are presented. Three of them conducted in conso-nances with real-world problems reported from industry; and

Chapter 6: a broad discussion about feature-based test oracles and the test of complex-output systems is provided. Finally, conclusions and final remarks on this dissertation are presented.

BACKGROUND

2.1 Overview

The central point of this dissertation refers to the mechanism for defining whether the current results agree with the expected outcomes in software testing scenarios – test oracles. Further, the test oracles proposed and evaluated in this research address to complex-output systems. This chapter presents concepts about software testing (Sec 2.2), test oracles (Sec 2.3), and complex-output systems (Sec. 2.4). Further, regarding each topic, we present details and running examples of automated software testing activities/tools, taxonomies of test oracles, and a brief discussion on contemporary complex-output systems.

2.2 Software testing

Despite the advanced resources of software development available, programmers can make mistakes during the process of writing code. Such mistakes set “faults” for performing a process, method, or activity. Manifestations of faults can lead software systems to inconsistent states and then several “errors” might be revealed (MYERS; SANDLER; BADGETT, 2011). In essence, the execution of a defective source code leads the program to an unexpected state and that is visible by means of the system output, revealing failures (IEEE, 2010; MATHUR, 2007). Software testing is one of the most effective way to identify faulty systems.

The current scenario in software quality reveals that there is no space for poorly tested software systems. Despite significant improvements of software development resources, testing activities have become increasingly important and crucial to the daily life of common people. Software systems developed within the last five years reach more people in a shorter time than decades ago, so delivering non-tested software might decree the unacceptability of a product (AMMANN; OFFUTT, 2008a). Demanding customers, ranking in software centers, customer’s reviews, etc. drive the current software systems to be literally viewed as a product and not a

solution to be chosen by consumers. These aspects increase the software industry competitiveness. Therefore, developers regard the test as one of the most important activities during the software development process. A conceptual background on software testing is presented next.

2.2.1 Concepts and theories

Myers, Sandler e Badgett (2011) emphasized that software testing activities can be the process, or series of processes, whose goal is running a system in order to find errors. According to Bertolino (2007b), software testing is the execution process of a software product aiming to check whether it reaches its specifications considering the environment in which it was designed. Hunter e Strooper (2001) agree that software testing is the main resource to check specifications against current behaviors. The standard way of thinking about software testing is that there are different useful definitions to describe it. In this study, we define software testing as a way to verify whether the SUT behaves in accordance with its specification through a controlled execution. Furthermore, Bertolino (2003) considers the following software testing definition as a reference framework, allowing different associations with testing concepts:

“... Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite execution domain, against the specified expected behavior ...”

Over more than 30 years of research and practice, software testing environments have suffered significant changes and advances (AMMANN; OFFUTT, 2008b). New software testing activities incorporate innovations in order to follow trends such as *Object-Oriented* (OO) (CHEN; TSE, 2013), *Aspect-Oriented* (AO) (FERRARI; MALDONADO; RASHID, 2008; CAPEO; MASIERO, 2011; KUMAR; SHARMA; GARG, 2009), web applications (PRAPHAMONTRIPONG, 2012; MCMASTER; YUAN, 2012; LEOTTA *et al.*, 2013), and embedded systems (YIN; LIU; NI, 2012; LATOUI; DJAHILI, 2013). This would allow one to affirm that test activities are highly dependent on the SUT. As a consequence of this dependence and particularity of the SUTs, many software companies allocate a specific team of practitioners to develop their own testing tools (BARESI; YOUNG, 2001).

Despite particularities, testing activities have commonalities, in the form of techniques and criteria that aim to mitigate efforts to discover errors and to decrease testing costs (MATHUR, 2007). Testing techniques and criteria promote testing activities in a systematic and judicious way. Each technique has the support of various test criteria to perform the exercise of different features of the SUT (MYERS; SANDLER; BADGETT, 2011; BERTOLINO, 2003). Therefore, testing techniques can be considered strategies to minimize effort and maximize efficiency for detecting defects (MATHUR, 2007). Regarding only testing techniques, one can highlight mainly “Functional testing”, “Structural testing” and “Error-based testing” (AMMANN; OFFUTT, 2008b).

Conventional wisdom has it that “Functional testing”, also known as black-box testing, consists of a testing strategy based only on the program’s specifications (MYERS; SANDLER; BADGETT, 2011). This testing strategy disregards SUT’s internal structures and it can be carried out using a set of test cases. On the other hand, “Structural testing”, also called white-box testing, employs SUT’s internal structures as a source of information for different tests and coverage analysis (DELAMARO; MALDONADO; JINO, 2016). Finally, “Error-based testing” uses information about frequent errors that are common during the process of writing code in software development projects (AMMANN; OFFUTT, 2008b).

In addition, during software development processes there are testing phases to be considered. Among these phases the most popular is: unit testing, which is the phase where the main test focus is to assess the correctness of the smaller units of the SUT (e.g: functions and modules); integration testing, whose goal is to verify the adequacy of different software modules working together as whole solution; and system testing which aims at detecting faults in systems (MATHUR, 2007; MYERS; SANDLER; BADGETT, 2011; BERTOLINO, 2003).

Applying appropriate software testing techniques and criteria during all phases of the development process, regarding the whole software life-cycle, imply a general increase of project costs for developers and, consequently, customers (MATHUR, 2007). However, according to Seo e Choi (2007), complexity and costs of software testing increase as do the size of the SUT. Consequently, at every stage of the development life cycle, it is necessary to apply testing techniques and strategies. Thus, many times, testing activities take more than 50% of the time of development (MYERS; SANDLER; BADGETT, 2011). For instance, it is possible to mention that a proper test for an SUT as a whole would take enough time to delay official commitments and deadlines. Therefore, different software companies have a policy of releasing unfinished systems under development and waiting for complaints from users to improve the system’s quality and reliability (SUN; LI; XU, 2012; IMPERATORE; SALVADORI; CHLAMTAC, 2007). It is often said that software developers and testers, generally, do not have enough time for testing their projects.

The most appropriate solution to alleviate the problems of cost and time associated with software testing is using automated testing activities during different stages of the development process and in different modules of the SUT (BERTOLINO, 2007b). An adequate automated testing has to provide productive and reliable approaches (IVORY; HEARST, 2001). Test automation is a matter of emphasis on the SE area because they promote more systematic testing approaches (IVORY; HEARST, 2001; BERTOLINO, 2007b). Automated testing activities can bring significant cost savings to software development projects, and is a key factor to help reducing analysis costs (AMMANN; OFFUTT, 2008b). Next we present common practical aspects associated with the automated testing in the Software Engineering context.

2.2.2 Automated testing

Despite the variety of techniques and criteria, the definition and proper usage of adequate testing strategies is a hard task. In real-world software development processes, the tester deals with at least four different delicate issues: *(i)* defining what aspects of the SUT should be tested; *(ii)* defining what are acceptable outputs and under which conditions; *(iii)* establishing test oracle configurations; *(iv)* identifying reliable sources of information and models which one can trust as a reference for test oracles.

Identifying the correct aspects related to such issues is essential for testers. Once testers identify all of those aspects, it is possible to adjust an adequate and automated test scenario, including a mechanism to support their final decisions (an oracle). The software product can be evaluated in a productive and effective way. Then, testers can load and run test cases automatically, without the necessity of human intervention after setting test cases, allowing automatic setting of system conditions and persistence of these conditions. This automated test scenario must include resources for comparisons between current and expected results. In other words, methods to analyze each individual test case result (pass/fail).

Due to the plurality of software systems available in the last years, it is possible to determine different meanings for the term automated testing. According to (DUSTIN; GARRETT; GAUF, 2009), an automated test may mean a wide variety of aspects associated with the testing processes such as, using capture/record/playback tools (NAGOWAH; DOORGAH, 2012), test-driven development (KARAMAT; JAMIL, 2006; MARKOVIC *et al.*, 2011), unit testing (RAMLER; MOSER; PICHLER, 2016; SHAMSHIRI *et al.*, 2015), custom scripts, etc. In spite of these variations, the aforementioned study highlights four fundamental characteristics that differs between manual and automated testing approaches: (1) automated tests are software development; (2) test automation enhances manual efforts once its focus is on automating hard manual tests; (3) automated testing activities do not replace manual tests and human intervention with testing strategies and techniques; and (4) automated tests cannot be separated from manual tests because these approaches are complementary.

To figure out an adequate bird-eye view of automated software testing, we list some basic conditions that should be achieved in a testing scenario. Some of these conditions were suggested by (HOFFMAN, 2001). According to this study, all conditions need not necessarily be satisfied to set an automated testing environment. These conditions are:

- Loading and running test cases automatically;
- Lacking human intervention after setting test cases;
- Allowing automatic system conditions setting and persistence of these conditions;
- Including resources for getting relevant results;

- Including resources for comparisons between current and expected results; and
- Including methods to analyze each individual test case result (pass/fail).

On the other hand, for the manual test, a human (the tester) provides inputs to the SUT and manually verifies its outputs. Manual testing involves six steps that do not necessarily need to be performed completely (MYERS; SANDLER; BADGETT, 2011; HUMBLE; FARLEY, 2010): (1) coding; (2) code update in some excerpts; (3) build/compilation; (4) running code and, sometimes, filling out test forms; (5) checking log files, databases, external services, screen outputs, variable's values, etc; (6) tester/developer results checking. In this scenario, manual approaches have become a common practice among software developers before they deliver a version for a testing team. In case the code does not work as it is expected to, some previous steps should be reconsidered (BERTOLINO, 2007b; HOFFMAN, 2006).

Table 1 presents basic comparative characteristics regarding automated and manual testing environments. These characteristics should be fairly presented pondering important aspects of a development and test environment such as cost and productivity (HOFFMAN, 2006). The advantages of adopting automated testing are worthy: simple replay, regression support¹, time to market and high productivity are some of them (BERTOLINO, 2007b; HARMAN *et al.*, 2013). On the other hand, manual testing is simple to adopt and has a low cost in the short term (HOFFMAN, 2006).

Table 1 – Implications for Manual and Automated Testing.

	manual	automated
test conduction	human-based	machine-based
re-execution	time-consuming	quick
regression support	no	yes
coding	simple	complex
project time	time-consuming	short
productivity	low	high

Source: Elaborated by the author.

There are many deep and fair comparisons between manual and automated testing regarding variations in project and team size. Using this thinking as a premise, Table 2 draws a comparison where advantages and disadvantages are numbered. In this comparison we consider several aspects such as, complexity, efficiency, efficacy, productivity, and cost. For the most of these aspects, automated approaches work better than manual ones. Although it is true that human monitoring is necessary and imperative for a wide variety of test scenarios. The most notable advantage of manual approaches is the fact that a person running tests manually can identify unexpected behaviors that automated testing might not be able to identify. A human

¹ Regression test should regard outputs from previous version of the SUT and use these outputs as expected outputs in recent versions of this same SUT

being is able to notice screen oscillations, delays or any unexpected behavior that an automated testing system would have difficulty identifying (HOFFMAN, 2006; HOFFMAN, 2001)

Table 2 – Comparison Between Manual and Automated Testing.

Manual testing	
Advantages	Disadvantages
<ul style="list-style-type: none"> • a human being monitoring might notice unexpected behaviors • notification of inconsistencies regarding different executions of data, states, settings and environments 	<ul style="list-style-type: none"> • efficiency in detecting errors must consider human fatigue • determined test data, iterations and combinations may require automation • certain aspects may not have manual solutions (i.e. performance analysis) • when testers are waiting for certain errors, after a few repetitions, this may impair the detection of defects
Automated testing	
Advantages	Disadvantages
<ul style="list-style-type: none"> • more complex, efficient and well elaborated test cases • tools are able to match amount, productivity, and efficiency • possibility of technology transfer between industry and academia • efficiency for error detections in specific domains 	<ul style="list-style-type: none"> • some complex domains of test provide extreme difficulty for automations • only check the conditions for which the automation was designed to investigate • difficulties on defining which conditions should be checked to consider a test as successful • implementations to generate efficient test cases are complex • difficulty in predicting the way an SUT should behave and what consequences are expected after an execution

Source: Elaborated by the author.

2.2.3 Supporting tools

The productivity of the testing activities in software engineering processes is intimately associated with the availability of adequate tools (PRESSMAN, 2010). In the last decade, the organizations have been adopting multiple testing tools for different purposes. Regularly, software organizations explore several testing tools in the software life-cycle. Testing tools provide systematic and robust environments to apply specific testing techniques and criteria. Table 3 displays a (incomplete) catalog of tools to support testing activities in different purposes.

As it is possible to observe, there are few testing strategies that address to unconventional and sophisticated outputs. Much of available testing strategies are done under conventional testing aspects. This is one of the major limiting factors to establish systematic and adequate testing environments to contemporary systems. Due to this, the SE research community is dedicating efforts on defining automated SE activities to meet the industry demand. However, regarding software testing aspects, these efforts are not being enough.

Table 3 – Testing tools, frameworks for testing and their application

Testing tool	Description	License	Purpose
OSS	framework of recommendations	free	security testing
Havij	SQL injection tool	Proprietary	
Pitest	mutation testing for Java	Apache2	fault-based testing
Mujava	mutation testing for Java	Apache	
Major	mutation testing for Java	Creative	
Proteum	mutation testing for C	GNU GPL	
Proteum/AJ	mutation using AspectJ	GNU GPL	
Randoop	Java test data generation	MIT	automated test generation
CodePro Analytix	Java test data generation	Creative	
EvoSuite	Java test suite generation	LGPL	
TestLink	test management system	GNU GPL	test documentation
FindBugs	static analysis for Java	GNU GPL	static analysis
PMD	multilanguage code analyzer	BSD	
JUnit	assertions for Java	Eclipse public	test case automation
Robotium	Android test automation	Apache	
Jmeter	performance testing tool for Java	Apache	performance testing
SonarQube	inspection of code quality	GNU GPL	quality management
Emma	Java code coverage tool	CPL	code coverage
EclEmma	Emma plug-in for Eclipse	Eclipse public	
JaCoCo	measuring and reporting Java code coverage	Eclipse public	
JaBUTi	structural testing for Java	GNU LPGL	
Selenium	record/playback unit test for Web	Apache 2.0	web testing tool
Watir	Web Application Testing in Ruby	MIT	
WatiN	GUI tests for Web applications	Apache	
Sikuli	robot-based UI testing	MIT	GUI testing
GUITAR	component-based GUI testing	GNU LGPLv3	

Source: Elaborated by the author.

2.3 A holistic view of test oracles

A fundamental element in the automated testing processes is the “test oracle”, which represents a method (program, function, set of data or table of values, etc.) to verify whether the SUT behaves correctly on a particular execution (HOWDEN, 1978; BARESI; YOUNG, 2001; SHAHAMIRI; KADIR; HASHIM, 2009). This element plays a decisive role in software testing processes establishing a decision about the correctness of an SUT result (WEYUKER, 1982; DAVIS; WEYUKER, 1981; DAY; GANNON, 1985). According to (IVORY; HEARST, 2001), test oracles must support testers’ decision about the SUT behavior against an input, establishing which results/behaviors are acceptable. In this sense, test oracles are reliable sources of informa-

tion that testers can trust to decide the correct test results (MACHADO, 2000; TAKAHASHI, 2001; BIEMAN; YIN, 1992). In technical terms, a test oracle can be carried out in different ways: functions, assertions, processes, data, parallel programs, and others (HARMAN *et al.*, 2013). On the other hand, even a human being can play the role of test oracle deciding about the correctness of test results. Regardless of the different ways of implementation, oracles bear a fundamental responsibility for the whole testing process: they judge and report to testers whether the result of an execution is correct (HOFFMAN, 1998; HOFFMAN, 2001; BARESI; YOUNG, 2001).

Developing an adequate test oracle for particular test domains offers a number of challenges. First, oracles are generally separated from the SUT and, then, one needs complex pre/post conditions to automate them in testing environments (JALOTE, 2005). Second, developing test oracles involves controllability and observability issues (AMMANN; OFFUTT, 2008b). This means that often it is necessary to determine external mechanisms to control and observe the functionality of the SUT, controlling and observing different inputs' effects on SUT's outputs. In addition, there are cases in which the tester must expend a precious amount of project time verifying test outputs in order to define whether they are acceptable. Finally, depending on the complexity of the output, in some occasions it is not possible to establish a test oracle. In these cases, the tester could explore heuristics (statistical functions or models) to provide expected results, establishing an oracle assumption.

2.3.1 Definitions and concepts

A plurality of definitions and concepts can be associated with test oracles. According to Machado (2000) and Hamlet (1995), the definition of test oracles concerns decision procedures and strategies for interpreting and judging the test results. When a set of test cases $\mathbf{TC}$ is available, the tester might implement test oracles as an object comparator between test case outputs and SUT's current output. Regarding an SUT $\mathbf{P}$, one can consider an input domain $\mathbf{D}$, an output domain $\mathbf{X}$ and, then, $\mathbf{TC}$ should be a function for matching the inputs $\mathbf{D}' \subseteq \mathbf{D}$ and the outputs $\mathbf{X}' \subseteq \mathbf{X}$. In this scenario, $\mathbf{D}'$ corresponds to a valid input in the domain $\mathbf{D}$ and $\mathbf{dom}(\mathbf{TC}) = \mathbf{D}'$ denotes the domain of $\mathbf{TC}$. In this sense, $\mathbf{TC}(\mathbf{t})$ should be considered a valid output for each $\mathbf{t} \in \mathbf{dom}(\mathbf{TC})$. Then, $\mathbf{P}(\mathbf{t})$ represents the result of $\mathbf{P}$ executing with input $\mathbf{t}$. Then, one can define that the program $\mathbf{P}$ meets $\mathbf{TC}$ if and only if it meets it on all inputs in $\mathbf{dom}(\mathbf{TC})$. Therefore, considering that $\mathbf{TC}$ might have infinite correspondences, a finite set of test cases should be selected. Considering this scenario, a function $\mathbf{O}$ called oracle for $\mathbf{P}$ on $\mathbf{TC}$ if for all $\mathbf{t} \in \mathbf{D}$:

$$O(\mathbf{t}) = \begin{cases} true(pass) & P(\mathbf{t}) = TC(\mathbf{t}) \\ false(fail) & P(\mathbf{t}) \neq TC(\mathbf{t}) \\ true(pass) & \mathbf{t} \notin dom(TC) \end{cases}$$

Test oracles represent a mechanism to check test outputs. One can consider this short

definition as satisfactory, however real software testing scenarios and different test output domains provide ample evidence that this premise must be interpreted in different ways (CHAN; HO; TSE, 2007; OLIVEIRA; DELAMARO; NUNES, 2010; DELAMARO; NUNES; OLIVEIRA, 2013). Before presenting deeper definitions, it is necessary to consider test oracles as a piece inside the software testing context. This piece can be represented, for example, by a tester checking outputs and writing test reports. In this case, the oracle is called a “human oracle”. On the other hand, this piece can be totally automated by several processes able to judge the correctness of test outputs.

Regarding the automated testing scenario presented in Section 2.2.2, one can observe that it is possible to set a test environment in which an automated oracle is not included. In this scenario, generally, testers apply several techniques for achieving structural coverage and in generating test data. However, the SUT’s outputs or behaviors should be checked or evaluated by a human tester (human oracle). The efforts and project time expended by the tester on evaluating test outputs is known as the “human oracle cost” (MCMINN; STEVENSON; HARMAN, 2010). Human oracles are common in several situations such as, evaluating the effectiveness of new test techniques, using the tester know-how about the SUT behavior, detecting unexpected errors, etc.

In order to achieve productive software testing in large scales, testers should automate their test oracles to mitigate manual efforts. Disregarding manual approaches, oracles play a vital role in automated testing and their neglect can result in an unproductive test. To illustrate this, Figure 1 presents a wide concept map of software testing context and it illustrates test oracles as a supportive technology (DURELLI *et al.*, 2013). The concept map represents a general view of software testing matching sources of information, test phases, test techniques, testing criteria, processes, and test oracles. According to this figure, a test oracle is a software testing technology that can be associated with different processes and test techniques.

Figure 1 – A Concept Map of Software Testing

Source: Adapted from Durelli *et al.* (2013).

Figure 1 provides a concept from which it is possible to infer that test oracles are independent from any other software testing aspects. Regarding a more specific software test

environment, test oracles play a fundamental role as the conclusion of a series of processes that might include test case generation, test data selection, test sequencing, and test execution. In order to illustrate a scenario like this, Machado (2000) suggested a generic model of a black-box testing process including test case generation, test data selection, automated execution, and test oracle. Figure 2 provides a practical example of a generic association among all of these phases in a practical scenario.

Figure 2 – A Generic Model of a Black-box Testing Process.

Source: Adapted from Machado (2000).

In Figure 2, depending on the output domain, a tester can implement a productive test oracle in the form of a program, a set of assertions, a function, a heuristic or a data structure with expected outputs. Addressing these varieties, according to Chan e Tse (2013), under practical and theoretical points of view, test oracles can frame fundamental differences. Concerning the theory, Chan e Tse (2013) argue by means of the program's specifications one is able to determine appropriate test oracles. However, it is important to highlight that, in practice, oracle mechanisms could be costly, hard to implement or may not exist (WEYUKER, 1982; HARMAN *et al.*, 2013; ASRAFI; LIU; KUO, 2011). So, the implementation of an automated mechanism to the same end as a human being judging test outputs is a challenging task to be faced by test designers (STAATS; WHALEN; HEIMDAHL, 2011).

Despite the variety, in practice, aspects related to test oracles deal with a primary issue in software testing activities – deciding about the correctness of an SUT **P**, against pre-determined test data. Figure 3 presents a traditional and generic test oracle structure using flowchart elements. In this case, the test oracle accesses the set of data needed to evaluate the correctness of the test output. This set of data comes from the specification of the SUT and contains sufficient information for supporting the oracle's final decision.

Figure 4 introduces several structures which represent test oracle functions. In the figure it is possible to notice that, in order to adequately play their rules, some test oracles may require test cases. However, there are cases in which test oracles are able to provide a test result based on test data (inputs). There are cases where a set of information and data about the expected results are needed to decide about the correctness of SUTs (Figure 4a) (GUDERLEI

Figure 3 – Generic Test Oracle Structure

Source: Elaborated by the author.

Figure 4 – Test Oracles Using Different Sources of Information

Source: Elaborated by the author.

et al., 2008; HOFFMAN; STROOPER, 1991; JANSSEN; ABREU; GEMUND, 2009; KIM-PARK; RIVA; TUYA, 2010; MCMINN; SHAHBAZ; STEVENSON, 2012; STAATS; GAY; HEIMDAHL, 2012). On the other hand, test designers can create test oracles from formal methods or specifications (Figure 4b) (AICHERNIG *et al.*, 2009; ANDREWS; ZHANG, 2003; CHEN; SUBRAMANIAM, 2002; D'SOUZA; GOPINATHAN, 2006; GIANNAKOPOULOU *et al.*, 2011). Regarding most sophisticated cases, test oracles can place into service test data inputs to derive expected outputs of the SUT (Figure 4c) (SANGWAN; BHATIA; SINGH, 2011; SHAHAMIRI; KADIR; IBRAHIM, 2010b; SHAHAMIRI; KADIR; IBRAHIM, 2010a; VANMALI; LAST; KANDEL, 2002; RAUF *et al.*, 2010). Finally, as we already mentioned, one can consider the human oracles, where testers can use their own knowledge about the SUT to check if outputs are in accordance with specifications (Figure 4d) (AFSHAN; MCMINN; STEVENSON, 2013; HOOK; KELLY, 2009; MCMINN; STEVENSON; HARMAN, 2010; SNEED, 1988). Other possibilities could be contemplated, for example, a test oracle can be designed considering a reliable model of output from which basic features can be extracted to be compared with current outputs from the SUT (OLIVEIRA; DELAMARO; NUNES, 2010; DELAMARO; NUNES; OLIVEIRA, 2013).

Regarding complexity and costs, one can consider test oracles based on expected output/behaviors as unsophisticated and quick to implement. Regarding Figure 4a, a tester can deal with these oracles using one of the various frameworks that have come to be known collectively as an “*xUnit*” family. *xUnit* frameworks allow the unit testing in SUTs implemented in different programming languages (JAVED; STROOPER; WATSON, 2007). For example, in “*JUnit*” (TEAM, 2013), which is an *xUnit* framework for Java, testers can get different test oracles in their code using assertions, which mean a true/false statement placed in a program to check unit and partial results. Algorithm 1 presents an example of a test oracle using the *JUnit* framework. In this example, line of code 4 represents a test oracle to check the correctness of the method “boolean Store.checkByTitle(String)”.

Algorithm 1: Test Oracle Using an “*xUnit*” Framework.

```

1      public void testDVDInStore ( ) {
2      Store store = new Store ( ) ;
3      boolean result = store.checkByTitle ("The Godfather III") ;
4      assertEquals (true , result) ;
5      }
```

According to Figure 4b, when testers have at their disposal a mathematical model that faithfully represents the SUT such as, an FSM or a Petri net, it is possible to automate a test oracle from this model. Along the same lines of the example presented, regarding to Figure 4c, it is possible to set a test oracle using test data. For instance, through outputs given from parallel executions or using another versions of the SUT, a tester can build a test oracle to compare those outputs and current outputs. In these cases it is important to assume that the version used, called the reference program, meets all the specifications of the SUT. This sort of test oracle frequently uses the cases of a regression test and a mutation test.

2.3.2 The oracle problem

The decision about the correctness of an execution and, consequently, revelations of failures is the most essential aspect related to any testing activity, even the manual ones (SHAHAMIRI; KADIR; HASHIM, 2009; STAATS; WHALEN; HEIMDAHL, 2011). In a software testing environment, test oracles play this essential aspect, which is a hard-task because it is not trivial to find out a set of expected outputs for SUTs (CHAN; TSE, 2013). For instance, we can wonder about a situation in which the SUT **P** must figure out the value of π on a determined precision of decimal numbers. Unless it has another program **Q**, which is working properly to solve the same problem, it is impossible to determine if **P**’s outputs are right or not (WEYUKER, 1982). Another similar case is when **P** represents a non-deterministic program and **P(vk)** matches different possible outputs, all of them correct (CHAN; TSE, 2013; STAATS; WHALEN; HEIMDAHL, 2011). That means, the process of deciding about correctness of SUT’s is often much harder than it seems and involves finding solutions for different kind of problems (WEYUKER, 1982; DAVIS; WEYUKER, 1981; CHEN *et al.*, 2012). Problems like these have been frequent

in the software industry during the last three decades. Due to this, testers consider the task to determine reliable sources to judge all SUT's outputs a complex and non-trivial.

In addition to the false positives/negatives, the "oracle problem" is set in cases when, using practical means, it is impossible or too difficult to decide about the correctness of test cases and test outputs (WEYUKER, 1982). Along the same lines, we would define the oracle problem as the absence of oracle or cases when it is too expensive to apply the oracle. Due to this fact, the oracle problem is a fundamental challenge presented in the literature (DAVIS; WEYUKER, 1981; MACHADO, 2000; HOWDEN, 1978; MURPHY; SHEN; KAISER, 2009). This is one reason why testers classify the design of test oracle as a complex and cognitive activity rather than being a routine activity (CHAN; TSE, 2013; HAMLET, 1995). Even having a test oracle to support evaluations of test results, practitioners frequently identify unpredictable points about the use of oracles (MURPHY; SHEN; KAISER, 2009). Depending on the oracle, the following problems may occur:

- *False positives*: the test result fails while the SUT is working well. In a test scenario, a false negative is the failure to reject a false null hypothesis; and
- *False negatives*: the test result passes, however, some inconsistent state might not be checked. Regarding a test scenario, a false positive is when a test result incorrectly rejects a true null hypothesis.

Some research aim to alleviate the oracle problem using specific techniques. In these cases testing tools that are able to support testers' decisions about the correctness of an SUT execution (JIANG *et al.*, 2013; OLIVEIRA; DELAMARO; NUNES, 2010; DELAMARO; NUNES; OLIVEIRA, 2013). However, testers must also consider the possibility that oracles can often support them making wrong decisions. Depending on the SUT, it is extremely difficult to predict expected behaviors to be compared against current behaviors (DELAMARO; NUNES; OLIVEIRA, 2013; NARDI, 2013). Failures can come out under different situations which make checking the result complex or impossible to be performed (MEMON; XIE, 2004). The oracle problem can be challenging when SUT outputs are given in complex formats such as, images, sounds or virtual environments (DELAMARO; NUNES; OLIVEIRA, 2013).

2.3.3 Test oracle trade-offs

Dealing with test oracles brings some trade-offs. For testers, these situations involve losing project time and increasing costs in return for gaining quality aspects. Scientific studies explore the trade-off between efficiency in identifying failures in SUTs and complexity in developing oracles (MCDONALD; STROOPER, 1998; ROSE; POULDING, 2013; ZHANG *et al.*, 2009). The more effective the oracle, the more complex the information that it uses to

determine if the program execution is in line with the expected results. Consequently, defining an oracle as complex as the original program is usually not acceptable.

Other trade-offs must be explored as well. For instance, if it is possible to define a fully automated oracle to identify all possible failures present in the SUT's outputs, there would be no need for the program itself, because the oracle could be used instead. Then, we could consider that, regardless of the goal in studying a new oracle approach, it is important to know: What already exists? Where does the new approach fit? And how do we compare them? This allows, for example, one to identify what position in the trade-off a new study fits. For practitioners, it becomes easier to identify what kind of approach is best suited for a particular problem, that is, which oracle approach should best be used to support the test of a software product.

Considering broader scenarios of test automation including test oracles, testers should anticipate dealing with more complex trade-off situations. In these cases, the trade-offs increase because an important aspect is that both the reliable test set problem (HOWDEN, 1976) and the oracle problem (WEYUKER, 1982) are open issues. A set of test data $\mathbf{T}$ for a software system $\mathbf{P}$ is reliable if it reveals that $\mathbf{P}$ contains an error whenever $\mathbf{P}$ is incorrect. According to (HOWDEN, 1976), an effective testing strategy which is reliable for all software systems cannot be implemented. In this context, there is often no way to define in practice a criterion test that, once applied, can identify all errors in every program; likewise, there is no way to define in practice a test oracle that identifies all wrong outputs in every program (DAVIS; WEYUKER, 1981). In particular cases, this means that one must consider existing resources, the number of possible inputs (input domain), complexity in defining the respective outputs for all inputs, time, and cost. Thus, the tester must always weigh the resources available with time and test effectiveness.

A lot of research on software testing focuses on approaches that seek to change the balance of the trade-off in favor of some variable related to it. In the case of reliable test set problems, for example, many studies aim to find an approach or data selection criteria to identify more errors with a fixed and finite amount of resources for an application domain (SRINIVASAN; LEVESON, 2002; STAATS; GAY; HEIMDAHL, 2012). Other areas of research involve the generalization of results, i.e., whether an approach for a specific domain can be used in other domains, and how effective it is (TORSEL, 2011; VAZQUEZ *et al.*, 2013). Comparative studies are often published to answer such questions (SHAHAMIRI; KADIR; HASHIM, 2009; AGARWAL *et al.*, 2012). Next we bring explanations about the most studied and reported test oracle taxonomies in accordance with researchers and practitioners.

2.3.4 Taxonomies

Different authors (BEIZER, 1990; HOFFMAN, 1998; BARESI; YOUNG, 2001; HOFFMAN, 2006; HOFFMAN, 2001; STAATS; WHALEN; HEIMDAHL, 2011; HARMAN *et al.*, 2013) have dedicated their efforts in writing theories and classifications and taxonomies of test

oracles. Due to the diversity of software domains, currently there is no standardized test oracle taxonomy and the existing classifications could depend on the source of information, the SUT's output or the automation method considered by the oracle. In this section we present some of the most common oracle taxonomies and classifications. We present oracle taxonomies found in different kinds of SE studies conducted by academia members and practitioners. In this section we separated these classifications into three groups: (1) generic taxonomies (Sec. 2.3.4.1); (2) specific taxonomies (Sec. 2.3.4.2); and (3) taxonomy using oracle information characteristics (Sec. 2.3.4.3). In addition, we present some example to support theoretical descriptions.

2.3.4.1 Generic classification

In this section we introduce general classifications in which test oracles can or cannot fit. Although these classifications are well known by many researchers and practitioners, they are not often mentioned in studies of the test oracle. These classifications are associated with the general purpose of the test oracles and they are not associated with technical aspects applied in order to implement the oracle function such as, source of information or automation level.

Pseudo-oracles and Partial Oracles (Davis e Weyuker (1981) and Weyuker (1982)): Davis e Weyuker (1981) were two of the first authors to formalize and discuss the oracle problem. They sought solutions that alleviate the fact that it is impossible in practice to define a complete and totally reliable oracle for all SUTs. Given the impossibility of defining an ideal oracle, the authors discuss two possible options: “pseudo-oracles” and “partial oracles”.

According to Davis e Weyuker (1981), pseudo-oracles are programs written by a second team, in parallel to the SUT, and following the same specifications. Both, oracle and SUT run with the same input data and outputs are compared. This concept can be considered in two aspects. First, one could consider any code or executable model as a pseudo-oracle, Second, the margin of accuracy between the pseudo-oracle and SUT could be applied to allow acceptable discrepancies between results.

There is no guarantee that an oracle is free of faults hence “pseudo”. Accordingly, when the SUT's output is not equivalent to the oracle output, one must go through the debug process to check which one actually has the fault. As we mentioned before, if the oracle is incorrect, i.e., the oracle indicates a failure when there is no failure, this is a false negative. On the other hand, if both programs present the same wrong result, the verdict will be that the SUT passed the test when it actually did not – false positive.

Along the same lines, partial oracles aim to identify where test results are incorrect, even without knowledge of the correct output (WEYUKER, 1982). In other words, a partial oracle should support the tester's decision about the correctness of a test execution without expected outputs. Testers might resort to partial oracles when it is impossible or too difficult to define the expected result. That is, they must account for what is certainly wrong and analyze whether the SUT's output is plausible. For instance, one can affirm that a result of a `sin` function cannot

be outside the “-1” and “1” boundary. If it is outside such a range, the result is clearly wrong. Examples of partial oracles are found in many different ways (RAN *et al.*, 2009; HUNTER; STROOPER, 2001; JUST; SCHWEIGGERT, 2010). Pre- and post-conditions often define contracts which must be respected in the code. Although it is true that if an SUT passes a test and no error is found, it may still be incorrect because the partial oracle only indicates that the program produces data within a plausible result, but it does not mean the result is confidently correct.

Passive and Active Oracles: Along the same lines of the classification presented before, the literature presents a generic classification among test oracles: passive and active oracles. It is often said that active oracles directly drive their own testing activities reproducing the SUT behavior and generating expected outputs (MCDONALD; HOFFMAN; STROOPER, 1998; MCDONALD; STROOPER; HOFFMAN, 2003; MILLER; STROOPER, 2003; SHUKLA; STROOPER; CARRINGTON, 2005). Contrary to this premise, passive oracles act as simple comparators between current and expected test results. According to Pasala *et al.* (2007), an active oracle mimics the behavior of the SUT and a passive oracle verifies the SUT behavior, but it does not reproduce it.

2.3.4.2 Specific Taxonomies

The general purpose of this section describes specific test oracle taxonomies on software testing presented in the literature. Research on test oracles defined the majority of these classifications regarding different aspects such as source of information, automation process, information characteristics, etc. In this section we present these classifications highlighting their particularities. Each topic represents a particular classification and the context in which this classification was suggested. We use a broad variety of practical examples to support our explanations.

Source of information for Oracles (Beizer (1990)): Beizer (1990), interested in characteristics used as sources of information in different test oracles, identified five different types of oracles. According to him, oracles are defined following different aspects such as, source of information, data input, generation of expected outputs, etc. Beizer (1990) then defined five types of oracles according to their sources of information:

- *Input/outcome oracle:* more complex and the most common, this test oracle specifies the expected outcome for a specified input;
- *Kiddie testing:* the principle of these oracles is common; it runs the test using test data and reporting whether the application crashes. In more complex cases, this type of oracle should verify some characteristics searching for incoherency, and after this, it gives a verdict;

- *Regression test*: expected outputs are given from the previous version of the SUT. Besides having a new version of the SUT, the tester should consider test data with the same outcome as the last version;
- *Pattern test set*: decisions are based on a pattern previously created and validated. This type of oracle is common for validating compilers, Web browsers, and word processors; and
- *Existing programs*: the test is performed based on the results of entries executed on an existing and similar system.

Classification by Automation Process (Hoffman (1998), Hoffman (2001), Hoffman (2006)): According to Hoffman (2001), besides serving as a reference for comparison of current outputs with expected results, the expression test oracle could be used to describe the generation of expected results (data). Then, disregarding manual approaches, behind every test oracle exists an automation process. In three different studies (HOFFMAN, 2006; HOFFMAN, 2001; HOFFMAN, 1998), the researcher *Douglas Hoffman* described several characteristics useful to the process of oracle automation. In the same studies, Hoffman presents a different oracle taxonomy. In this context, we compiled these characteristics defining a taxonomy based on the automation processes described by Hoffman. This taxonomy is presented in Table 4. The table shows the oracle classification, expressive features, advantages, and disadvantages classified into different classes.

Oracle Categories According to Harman et al. (2013), Barr et al. (2015): In a recent scientific study, Harman et al. (2013), Barr et al. (2015) provide a broad analysis of trends in research on test oracles regarding current approaches. In their analysis, the authors considered the works on test oracles into four different categories:

- *Specified oracles*: in this category Harman et al. (2013), Barr et al. (2015) include all test oracle that specifications in order to judge test outputs and behaviors. Naturally, a lot of different studies could be included in this category. Among these studies, the authors highlight specification based languages, models, assertions, contracts, and algebraic specification;
- *Derived oracles*: in this group of test oracles, tests included approaches using any type of artifacts or resources from which test oracles may be created. Given this definition, the authors highlight that derived oracles may become an uncompleted specified oracle. The common resources cited are MR, regression test suites, system executions (trace), textual documentation, and strategies for pseudo-oracles;
- *Implicit oracles*: are intended to identify situations in which the presence of a fault in the SUT is obvious such as, crashes. Harman et al. (2013) highlight that implementing

Table 4 – Compilation of Testing Oracles by Their Automation Process.

Oracles	Definition	Advantage	Disadvantage
No oracle	Approach without verification of results	Large amount of data can be executed	Only notable errors are detected
Human	A human figure verify results	Can detect unpredictable errors	Time consuming and its efficiency is influenced by physiological factors, such as, tiredness
True or Complete	The oracle generates outputs from inputs	All possible errors, considering the test set, are detected	Expensive implementation. Its execution is time consuming
Consistency	It compares current results to previous results (regression testing)	Quick verification. Can verify large amount of data	Does not detect errors in the original program
Stochastic or Random	Checks a random test data sample	Could automate testing in a simple manner	May not notice specific errors and it is time consuming to check
Sampling	Verifies a previously selected test data sample	Generally promotes a quick check	May not detect specific errors
self-referenced	Promotes responses to data via messaging	Allows wide post-test analysis. The verdict is given in the message content	The system must submit test results so that messages are received
Heuristic	Verify specific characteristics	Quick, simple and inexpensive	Generally could bring false positive/negative results
Model-based	Uses a digital model data about the behavior of the SUT	Tests can be used across multiple systems using different models	Maintenance is expensive. Complex models must match the expected behavior
Manual	Results should be carefully defined by a test engineer	Useful for complex SUTs	It always performs the same procedures and is limited by a number of test cases
Statistics	Makes statistical correlations between inputs and outputs	Enables checking large scale and real data systems	May not detect apparent errors
Computational	Explores the behavior of the SUT to convert inputs into results	Very useful for simple mathematical functions and transformations	May not detect some errors and generate false positive/negative results

Source: Elaborated by the author.

these oracles requires no domain knowledge and the technique can be used in all runtime problems. A function which is always waiting for handles of exceptions is an example of this test oracle;

- *No oracles*: in this group of study, the authors include some approaches on test oracles which try to deal with the test oracle as a whole. In these studies, despite the absence

of automated oracles, the researchers' target is generally to reduce the human efforts on the judgment of test outputs. Among these targets, the authors highlight reducing human oracle cost, reducing qualitative human oracle cost, and crowdsourcing the oracle.

2.3.4.3 A Taxonomy by Oracle Information Characteristics ([OLIVEIRA; KANEWALA; NARDI, 2014](#))

Oracle information is a standardized term by researchers of test oracle use in order to reference data about expected test outputs ([NARDI; DELAMARO, 2011](#)). Then, oracle information represents the SUT's expected behavior ([MEMON; BANERJEE; NAGARAJAN, 2003](#)), which might be obtained from the specification, stored results, parallel program execution, learning machines, and other sources. The oracle information can be concrete (that is, the expected result itself), or abstract, as the acceptable boundary of results, expressed by a constraint. Complementing the concept of oracle information, oracle procedure is another standardized term used by researchers and it represents the processes used to compare the oracle information with the current output. According to [Durrieu, Waeselynck e Wiels \(2009\)](#), systems perform this comparison at runtime (online) or after the execution (offline).

The source of information and the way this information is represented, can influence the accuracy, complexity, expressiveness and many other aspects of the oracle's effectiveness. Furthermore, the sources of information are critical components to determine the ease of writing and interpreting their meaning. In [Oliveira, Kanewala e Nardi \(2014\)](#), we suggest an oracle classification regarding the oracle information explored by studies on test oracles. This taxonomy assumes that studies on test oracles fit into four basic types: (1) *specification-based* oracles; (2) *metamorphic relation-based* oracles; (3) *machine learning* oracles; and (4) *version-based* oracles. Disregarding a few differences, our taxonomy is quite similar to classification suggested by [Harman et al. \(2013\)](#), [Barr et al. \(2015\)](#).

Specification-based Oracles ([Baresi e Young \(2001\)](#)): [Baresi e Young \(2001\)](#) present a specific survey highlighting the main characteristics of automated test oracles that require neither pre-computed input/output pairs nor a previous version of the SUT. Consequently, regardless pre-computed outputs, software specifications are the most important source of information to derive test inputs and test outputs. This category of test oracle can be named specification-based. Due to the popularity of this test oracle approach, this section brings several examples of differences forms to implement test oracles using SUT's specifications.

Technically, a specification can be defined as a detailed formulation, in document form, which provides a definitive description of a system for the purpose of developing or validating the system ([IEEE, 2010](#)). When this formulation is used for validation, it is regarded as the oracle information. The language used to describe such a formulation is called a specification language, which is capable of being interpreted by a compiler or interpreter and allows the creation of an automated oracle procedure.

The usage of specification languages as oracle information is also supported by a variety of studies found in the literature. Specification-based oracles can be classified according to the paradigm, automation features, levels of abstraction, and other characteristics. Here, we present specification-based oracles regarding three general aspects: (1) their location in relation to the SUT; (2) their paradigm; and (3) their temporal property representation.

Specification Location: Baresi e Young (2001) present groups of oracles according to certain similarities, among them: oracles of pure specification language, embedded assertion languages and extrinsic interface contracts. These three oracle contracts differ from each other in the way they are written with respect to the code.

Oracles based on “pure specification” are those in which the tester uses a specification language to describe the desired behavior of a system or part of it for later use as a source for the oracle, i.e., as the oracle information. Such languages are usually not designed with the concern of being automatically interpreted. Therefore, defining procedures to interpret oracles which use such a source of information can be challenging.

“Embedded assertion languages”² allow one to insert expressions of intent within the code to be tested. They usually represent pre- or post-conditions at some control point in the program. Such expressions are checked during the code execution and, if a violation is detected, a message is presented. In this way, they are executable code which may be an extension of the same language used to implement the SUT or from a different language.

Java and other programming languages natively support embedded assertions. There are also specification notation languages, such as Anna (LUCKHAM; HENKE, 1985; SANKAR; GOYAL; SIKCHI, 1993; HAGAR; BIEMAN, 1996) and *Java Modeling Language* (JML) (CHEON; LEAVENS, 2002; MURPHY; KAISER, 2009), in which the assertions can be written in the code. In such cases, assertions are marked with reserved words recognized by an outside interpreter/compiler and executed as a separated part of the code.

For example, consider a Java code snippet which was modified to increase its performance, where the tester wants to be sure that the output is the same as its original output. In this context, one may introduce an assertion in the SUT to ensure that an error will occur if the refactoring method for coordinate calculation differs from its original. As an illustration, in Figure 2, the first line contains the embedded assertion recognized by the Java compiler and, consequently, *Java Virtual Machine* (JVM) given the reserved word “assert”. During the program execution, if the expression is not true, a runtime error is thrown. Technically, this is the oracle

² Assertion may be defined as a logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution (IEEE, 2010)

strategy explored by *JUnit*.

Algorithm 2: Excerpt from an Assertion Code.

```

1  assert (calculate_coord() = newCalculate_Coord());
2  float new_coordinate = newCalculate_Coord(coordinate);

```

“Extrinsic Interface Contracts” keep the specification, in the form of assertions, separate from the implementation and less closely tied to the target programming language. There are several assertion languages that allow extrinsic interface contracts, known as *Architecture Description Language* (ADL). Developers must define the bindings between the specification and the functions in the program (DAI *et al.*, 2007; HARMAN *et al.*, 2010). This is usually achieved by the use of wrappers. A wrapper is a checker that surrounds the component under test (EDWARDS, 2001) without modifying its code. For instance, when testing a given class on an OO code, another class (the wrapper) is created with the same interface as the original but with other methods which are responsible to check some constraint.

Figure 3 presents a code example, adapted from Shukla, Strooper e Carrington (2005), in which the objective is testing a class which is a list of integers, using an insert method. The code represents the list class to be tested. A list of integers, in this program, should have a maximum length of 1,000 elements and three methods: `insert()`, `size()` and `exists()`. To test the `insert()` method, two basic assumptions are considered: (i) if a value is inserted, it must exist in the list; and (ii) if a value is inserted, the list must have one more element than before the value is inserted. Both assumptions can be tested without inserting intrusive code (embedded assertions) with a wrapper.

Algorithm 3: A Program to be Tested Using Interface Contracts

```

1      public class ListOfInteger{
2          public final int MAX_LENGTH=1000;
3          private int index=0;
4          public int[] list = new int[MAX_LENGTH];
5          public void insert(int value){
6              //The insertion code is here
7          }
8          public int size(){
9              return index;
10         }
11         public boolean exists(int value){
12             //the code to check if an element exists is here
13         }
14     }

```

Figure 4 represents such a wrapper. It extends the original class and overrides the method to be tested. The new `insert()` method (i) retrieves the size of the list before the insertion, (ii) calls the method under test, (iii) retrieves the size of the list after the insertion, and (iv) calls the method which will evaluate the assumptions. If the value is not inserted into the list or if the list does not contain one more element after an insertion, a message is sent. A driver calls the

method `insert()` from the wrapper and the class under test is evaluated without the need for intrusive code.

Algorithm 4: Code Representing Class to be Tested

```

1      public class Wrapper extends ListOfInteger{
2          public void insert(int value){
3              int before = size();
4              super.insert(value);
5              int after = size();
6              checkInsert(value, before, after);
7          }
8          void checkInsert(int value, int before, int after) {
9              if ((!super.exists(value)) || ((before+1) != after));
10                 System.out.println("Error on insert()");
11          }
12      }

```

Other specification languages may be related to the SUT by some variation of the presented approaches. For example, a specification does not need to be a contract, but can be any other language paradigm which can be translated by the oracle procedure. The mapping between oracle information and SUT can also be achieved by instrumenting the SUT to dump the target outputs to a log which can then be used by the oracle procedure in the analysis.

Specification Paradigm: There are several specification languages and their respective paradigms are reflected in many different oracle approaches. Because any programming language can actually be interpreted as oracle information, there are as many oracle information paradigms as language paradigms. Here we present some of the most common paradigms that can be found in the literature to write oracle information.

Embedded assertions, discussed in the last section, are “declarative languages” that are non-procedural languages that permit the user to declare a set of facts and to express queries or problems that use these facts (IEEE, 2010). Other examples of declarative languages are *Object Constraint Language* (OCL) and Alloy, which are not embedded assertions. Such oracles are examples of partial oracles.

OCL, for example, is an extension of *Unified Modeling Language* (UML) proposed by the *Object Management Group* (OMG). It allows the definition of constraint limits of values to variables and pre- and post-conditions of methods (PACKEVICUS; ANDREJ; BAREISA, 2007; BRIAND; LABICHE, 2002; BRIAND; LABICHE; SUN, 2003; CHEON; AVILA, 2010). An example of a study of automation is presented by Cheon e Avila (2010), in which OCL constraints are translated to runtime checks in AspectJ³ separated from the implementation code, and mapped into the SUT with pointcuts.

Other languages provide “procedural” or “object-oriented” resources, such as Object Z which allows the tester to plan his oracle information with such concepts as inheritance. An

³ *AspectJ* is an extension of a program paradigm known by AO Programming for the Java programming language.

external checker is needed to translate specification sentences, as previously stated. Perfect Developer (CROCKER, 2003) is a tool that incorporates an automated theorem to verify that specifications written with Object Z (or a specification language called Perfect) are sound. Such a tool is not an oracle in the sense that it does not compare the specification with the SUT. Rather, it analyzes whether the specification has flows that work as contradictions.

Another specification paradigm encompasses “executable models” which can be simulated in tools such as *Simulink*, *XCos*, and *Scicos*. They are suitable as oracle information because it is possible to simulate the SUT behavior. An oracle procedure should retrieve the SUT results and compare them with the model results. Such an oracle is a simplistic example of pseudo-oracles because it uses another version of the SUT to test it – the executable model (NARDI, 2013).

A practical example of the use of executable models as oracles is presented by Lasalle, Peureux e Guillet (2011). The authors apply a *Simulink* model as oracle information. Its simulation calculates the expected results of a vehicle reaction with respect to the road characteristics when steering.

State machines and other models can also be used as oracle specifications. A parser can be applied to generate an analyzer from the machine. The SUT’s output is inserted into the analyzer. After that, if the output is not expected in the machine, an error is detected. In this case, the state machine plays the role of the oracle information and the analyzer represents the oracle procedure.

Andrews e Zhang (2003) give an example of state machines as oracles to test an elevator system. In their model, the output is stored into a log file. The requirements are: (i) the door must be closed if the elevator is moving; (ii) the elevator must be stopped when the controller program terminates; and (iii) the door must never be open for more than 30 seconds. A sample of SUT’s output is presented in Figure 5.

Algorithm 5: A Sample of SUT’s output Regarding State Machines as Oracle.

```

1      call 3
2      go_up
3      reach 2
4      reach 3
5      stop
6      door_open 3 103325
7      door_close 3 103340
8      go_down

```

The testers have created two machines: one for requirements 1 and 2 (Figure 5 (door safe)), and another for requirement 3 (Figure 5 (door delay)).

An analyzer receives the following inputs: the state machine and the log file. If the analyzer cannot recognize a line from the log or if there is no transition in the machine, the file is rejected and an error is identified. Eventually, the analyzer may be modified to ignore valid

Figure 5 – State-Machines as Test Oracles

Source: Adapted from [Andrews e Zhang \(2003\)](#).

log lines that are not represented in the state machines. In the example, “*call*” and “*reach*” are expected lines in the log file, but they are not represented in the state machines. In this case, they must be ignored by the analyzer so as to not trigger an error.

Temporal Specifications: Some software systems must be in agreement with requirements that express temporal properties. This is common for embedded systems, as in Avionics and Telco. A requirement example of a Mars probe ([WANG; QI; LI, 2005](#)) is expressed as follows:

“... When the TimerInt reaches the Control System and the reading of acceleration is not completed, the status should change to Emergency within 60 milliseconds ...”

One can observe that a behavior (status changing) must be true within 60 milliseconds, that is, not necessarily at the same instant as the other triggering behaviors. The oracle must have the ability to check such requirements in the SUT. There are many specification languages from diverse paradigms which can be used to write temporal properties. TRIO ([LIN; HO, 2000](#); [FELDER; MORZENTI, 1992](#)), MITL ([WANG; QI; LI, 2005](#)), EAGLE, GIL, time Petri nets ([LIN; HO, 2001](#)), and timed automata ([WANG; QI; LI, 2005](#)) are other instances.

Higher expressive temporal languages should express time quantitatively, qualitatively and provide some representation of existential ($\exists$) and universal ($\forall$) quantifiers. Quantitative operators allow the representation of intervals of time in relation to other events. For example: “event X must hold until event Y happens”. Qualitative operators provide ways to measure distances, as “X must hold k units of time from the current instant”. For instance, [Felder e Morzenti \(1992\)](#) provide such operators.

As with pure specification language, the execution of a temporal language is not always the main concern. There are studies which tackle the adaptation of pure languages on automated testing tools which are capable of translation. Given the high complexity of such languages and

the difficulty of defining a complete solution, the adaptation is usually accomplished partially in exchange of some expressiveness.

This is the case with the research on oracles for Simulink-like models (NARDI, 2013), where a tool is capable of analyzing results with respect to a TRIO-adapted specification (TRIO/Apolom). Although a recursive capability is not implemented, the oracle procedure is capable of analyzing quantitative and qualitative operators, such as existential and universal quantifiers.

The following requirements are used as a base to write an oracle information example in TRIO/Apolom language:

“... If the temperature of a system is greater than x for an interval of time (interval) of more than $t1$ instants, then a red indicator must be on to indicate a critical situation until it lowers to a safer temperature. A safety protocol (safety) must be started within the next $t2$ instants and should last the same number of instants as interval. Also, a yellow indicator must be on after the safe temperature is reached and until the alarm is turned off ...”

Metamorphic Relation-based Oracles: the oracle information may not be based directly on the system specification, but rather on known relationships between multiple inputs and outputs, known as *Metamorphic Relations* (MR). MR specifies how the output of the program should change according to a specific change made to the input and represent some necessary properties of the (CHEN; TSE; ZHOU, 2003). MRs are used as test oracles in Metamorphic testing to identify faults in the SUT (CHEN; CHEUNG; YIU, 1998). Following are the basic steps in applying metamorphic testing:

1. Identify set of MRs that should be satisfied by the SUT;
2. Create set of test cases using a traditional test selection approach such as random testing or fault based testing. These test cases are referred as initial test cases or source test cases;
3. Create follow-up test cases by applying the input transformations specified by the MRs identified in step 1 to the initial test cases created in step 2; and
4. Execute the initial and follow-up test case pairs to check whether the output change at runtime complies with the output change specified by the MR.

As an illustration, the $\sin$ function must always obey the following property: $\sin(x) = \sin(180 - x)$. Such necessary properties of the $\sin$ function may not be a part of its specification. However, this property specifies a relationship between a pair of input data, x and $180 - x$, and their respective outputs $\sin(x)$ and $\sin(180 - x)$. In the given example, the follow-up test case can be created by subtracting 180 from the initial input. Testing of an implementation of $\sin$ can be performed by executing the following initial and follow-up test case pairs and checking whether the produced output pairs are equal:

(3, 177), (15, 165), (45, 135), (125, 55), (277, -97)

Although this is a trivial example, MR based oracles are applied for testing applications in different domains, including bioinformatics (CHEN *et al.*, 2009), machine learning programs (XIE *et al.*, 2011; MURPHY *et al.*, 2008), embedded software (KUO; CHEN; TAM, 2011), health care simulations (MURPHY *et al.*, 2011), Monte Carlo modeling (DING *et al.*, 2011), and programs with partial differential equations (CHEN; FENG; TSE, 2002). In addition, (XIE *et al.*, 2013) integrated MRs with program slicing and used that for fault localization in programs that face the oracle problem.

The set of MRs used in testing has a big impact on the effectiveness of metamorphic testing. Liu *et al.* (2014) found that using more diverse MRs can improve the effectiveness of testing (LIU *et al.*, 2014). Usually, MRs used for testing are identified manually by the tester, based on her knowledge of the program specification. Recently, there have been developments of automated methods that use machine learning techniques to detect MRs (KANEWALA; BIEMAN, 2013). These approaches use a set of features that represents the static control flow information of a program to develop machine learning prediction models that are used to predict MRs.

Some taxonomies on test oracles consider metamorphic relation-based oracles as a subcategory of derived oracles or specification-based oracles (HARMAN *et al.*, 2013). For the context of this chapter, due to the increase of approaches on metamorphic testing over the years, we have considered metamorphic approaches as their own category. We think that studies on metamorphic testing are mature enough to be analyzed and discussed separately.

Machine Learning Based Oracles *Machine Learning* (ML) is a set of computational methods that use collected data and is capable of making predictions. There are a variety of ML techniques applied as oracles, such as *Supervised Learning Machines* (SLA): (1) *Artificial Neural Networks* (ANN) (AGGARWAL *et al.*, 2004; ALMOG; HEART, 2010; JIN *et al.*, 2008; MAO *et al.*, 2006), (2) support vector machines (WANG *et al.*, 2011; WANG; YAO; WU, 2011), and (3) info-fuzzy networks (AGARWAL *et al.*, 2012; ZARAKET *et al.*, 2012).

These three SLAs share a characteristic: they use collected and labeled data to train a machine to predict new, unlabeled data. ANN, for example, can be trained with pairs of inputs and already known outputs (training set). The resulting machine may be seen as an approximated function of the SUT. If a new input is used in the trained ANN it is capable of predicting an approximated output, even if this input-output pair is not part of the training set. Therefore, the ANN can be used as an oracle. If the same input is used in the ANN and the SUT, a procedure can compare both and identify if there are irregular discrepancies.

A similar application of ML as oracles is the use of them as classifiers. In the same way, pairs of elements and the class or category in which they belong are used to train an ANN. At the end of the training step, the ANN may be capable of identifying in which category new elements

belong. For instance, [Aggarwal et al. \(2004\)](#) and [Jin et al. \(2008\)](#) present a case study with oracles for triangle classification into isosceles, scalene, equilateral or not a triangle. The ANN receives two inputs: a triplet which represents three sides of a triangle and the category in which it fits. After the ANN is trained, it is capable of predicting in which category new triplets belong.

Such an oracle approach is appealing because of the ability to approximate functions. However, it has limitations to overcome. Input data may not be easily represented as characters and strings for use in ML. Also, deciding the structure of the network, such as the number of layers and neurons, may not be easy. The selection of training sets from test cases is another key problem that must be considered carefully: it should first be evaluated, which requires the use of other oracles.

Version-based Oracles: Version-based oracles are mechanisms that explore different version of the SUT to support decisions about the correctness of a test execution. The use of other versions of the SUT to implement test oracles was previously discussed in the introduction of pseudo-oracle concepts. This classification is very similar to the Derived Oracles, presented by [Harman et al. \(2013\)](#). Here, we briefly discuss different approaches to SUT versions as oracle information and how new versions can be used, namely:

- *N-version*: an N-version approach requires that N independently written versions of an SUT are also implemented ([ECKHARDT D.E.; LEE, 1985](#)). If different outputs are produced by any version, a majority vote decides which output is likely correct. If the same team implements all versions of the SUT in the same language, it is possible that the same defect in the program will be present in other versions (correlated defects). To avoid such a drawback, [Manolache e Kourie \(2001\)](#) suggest that different teams and language paradigms should be used. However, [Knight e Leveson \(1986\)](#) found that even independently developed programs may contain correlated faults; and
- *M-model program testing (M-mp)*: an M-mp approach is a variation of the N-version. In this case, only different versions of SUT are implemented, instead of a complete system, with the objective of reducing the test cost.

Other similar approaches of version-based oracles may include three variation of strategies. First, “regression testing” in which previously tested stable versions are used as oracle information to test programs developed by iterative processes ([MEMON; BANERJEE; NAGARAJAN, 2003](#)). Second, “third-party components” that can use tools such as extreme harvesting ([HUMMEL; ATKINSON, 2005](#)) to search for artifacts (as metadata) produced during normal development processes as the basis for component retrieval and matching on Web search engines. The retrieved components are then used as an oracle, and information. Third, one can mention oracles based on “mutation test” that uses outputs taken from programs generated from the original code with small modified source code (mutant programs). Then, the oracle compares these outputs with outputs from the original code.

2.3.5 Mapping study and historical evolution

We have conducted a mapping study to catalog the a significant amount of test oracle studies. In the following section we present the methodology we have used to acquire a pool of 304 test oracle studies. We have created an online repository⁴ including all of these studies and important information such as title, authors, year and medium of publication, abstract, and others. After constructing the repository, we defined a series of data points to be collected from these studies in order to provide a bird's-eye view of the issue. In this section we explain our methods and protocols for research and present the numbers that we derived from our analyses.

2.3.5.1 Study selection strategy

We searched for studies on indexed databases aiming to obtain a significant number of useful studies on test oracles. We have conducted web searches for studies in *Institute of Electrical and Electronics Engineers* (IEEE)⁵ and *Association for Computing Machinery* (ACM)⁶ using the following search string:

```
"(("test" OR "testing") AND ("oracle" OR "oracles"))".
```

We searched for these terms in three different fields: title, abstract, and keywords. We did not customize the searches for specific years, journals or conferences. All of this information is important for possible replications and future updates. Further details can be found on [Oliveira, Kanewala e Nardi \(2014\)](#).

After making adjustments for each database, we collected more than 400 studies, combining ACM and IEEE results. After that, we read each paper's abstract, analyzed the studies (excluding repeated entries) and selected the ones related directly to test oracles. Studies on SE topics other than test oracles were excluded. On the other hand, any study describing, examining, or developing any kind of test oracle was included in our analysis. We avoided imposing many restrictions on study selection in order to get a broad publication overview.

Following this manual selection we added some already known primary studies related to this topic. These papers did not appear in our original pool because they had appeared in publications not indexed by the two selected databases. Then, at end of the process described, we gathered a pool of 304 studies directly related to test oracles in an automated software testing scenario. One can access online the complete list of these studies and information about them through the repository web page.

⁴ Repository available in: <http://www.labes.icmc.usp.br/~rpaes/repo/repo.html>

⁵ see: <http://ieeexplore.ieee.org/>

⁶ see: <http://dl.acm.org/>

2.3.5.2 Study classification

We have classified the selected studies according to the oracle taxonomy by oracle information characteristics, detailed in Section 2.3.4.3. This means that we have read each selected study, classifying them in four different categories: (i) specification based, (ii) metamorphic relation, (iii) machine-learning, and (iv) n-version or similar. During this analysis we found some secondary studies, which are studies that analyze other studies with new contributions and results (primary studies) (KITCHENHAM *et al.*, 2010). Then, we classified these sorts of studies as (v) surveys. In our survey we identified secondary studies with classifications or surveying test oracles aspects.

2.3.5.3 Result analysis

Analyzing our pool of studies, one can note that, in a generic way, the interest from research of test oracles grew during the years. It is possible to note that the amount of research about this field has increased year after year since 2008. Using a line graph, Figure 6 presents a visual analysis of the number of studies included in our pool from 1978 to 2013. In addition, the same figure presents a table including the number of studies identified each 3-year period from 1978 to 2013.

Figure 6 – Analysis of Published Research on Test Oracles.

Source: Elaborated by the author.

These quantitative aspects show a considerable evolution of research on test oracles. The upshot of all this analysis is that the oracle issue is growing year after year, receiving increasing attention from researchers. Research on test oracles has been quite constant throughout the underlying 25-year period, since 1987. Considering the selectivity of the conferences and

journals and its broad coverage, it is fair to assume that research on test oracles have been playing a substantial role in the SE context.

Our analysis and classification shows that research on test oracles have been published under the five distinct categories in the following proportion: specification based approaches are responsible for 71% (217/304) of our selected studies; MR-based account for 12% (36/304); N-version and related studies account for 6% (17/304); machine-learning approaches represent 7% (20/304) of the approaches considered; finally, surveys compose 5% (14/304) of the studies. Figure 7 draws a fan plot that illustrates the differences among the approaches considered in this study. In addition, the same Figure includes a table that presents these numbers sorted from the highest to the lowest.

Figure 7 – Pie Chart of Test Oracle Approaches

Source: Elaborated by the author.

The line graph in Figure 8 illustrates the relative quantities and differences among studies selected for our study. At a glance, specification-based approaches have been in a continuous evolution since 1995. Considering approaches related to MR, it is possible to note a considerable evolution after the year 2000. The evolution has established a regular constancy, showing that this area has been widely exploited by new research. Machine-learning approaches has maintained a certain constancy without many significant developments. Along those the same lines, the N-version and similar approaches appear as a supporting approach that did not show many evolution. Finally, as a consequence of the popularity of automated testing activities, surveys on test oracles are becoming common among other approaches.

Complementing the analysis on the evolution of software testing approaches, Figure 9 shows graphically the number of oracle publications during each three year period since 1978. Regarding this visual information one can notice that since about 2004 the four approaches were firm and they appear together on subsequent research. Using this analysis, one can infer that sketches of research on N-version based oracles were noted first at the end of the 1980s and,

Figure 8 – Year by Year Evolution of Oracle Approaches.

Source: Elaborated by the author.

after almost a 10-year period with no new results, these approaches came out again. Although it is true that after 2009 research has exploited more of these approaches. Finally, considering the visual information plot in this figure it is evident to note how much bigger the number of specification-based approaches when it is compared to other approaches.

Over 36 years of research, on new development systems on test oracles arose, as did APIs, programming paradigms, and languages with built-in tools. Regarding these changes, we accomplished an analysis to investigate test oracle research considering many systems implemented over these years. We have classified the SUTs in five distinct categories: (1) mobile/embedded or real-time reactive systems; (2) GUIs (*Graphical User Interface*); (3) general purpose; (4) Web and Internet applications (including SOA (*Service-Oriented Architecture*)); and (5) distributed and concurrent applications. Among general purpose we identified single functions and other popular SUTs and validation strategies such as, clock alarms, calculators, specific-domain tools, triangle classification, credit approval applications, ATMs (*Automated Teller Machine*) systems, cryptographic systems, numerical problems, and sort algorithms (bubble, quick, etc). Table 5 presents the number of SUTs per category identified in our analysis.

In addition to these information on test oracle, in the study published in [Oliveira, Kanewala e Nardi \(2014\)](#), we present a broad analysis on test oracles studies. Some of the information provided are: empirical evaluation analysis (toy versus real-world applications), studies origin (academia versus industry), demographic analysis (countries of universities and in-

Table 5 – Number of SUTs and Categories.

SUT category	#	%
(1) Mobile/embedded or real-time/reactive	44	17
(2) GUIs	15	6
(3) General purpose	164	62
(4) Web applications	35	13
(5) Distributed and concurrent	6	2

Source: Elaborated by the author.

Figure 9 – Comparative of Oracle Approaches Year by Year.

Source: Elaborated by the author.

dustries), venues and publication strategies, most prolific researchers, researchers' collaboration, surveys and position studies, supporting tools, and academic studies (Masters and Ph.Ds).

2.4 Complex-output systems

This dissertation's subject is test oracles to support testing activities and quality evaluation of complex-output systems. This section presents theoretical foundations about the association of complex-output systems and testing activities.

Systems' outputs correspond to the form of producing information that users consume. A software system processes and generates these information from an input (SHU; BAI; PODGURSKI, 2011). In a pragmatic way, software systems produce outputs following a threefold flow that consists of (i) consuming some *input*; (ii) *processing* this information following some rule; and (iii) producing an *output*. This workflow is commonly known as *input–process–output* (IPO). Once this workflow is done, software output can assume different domains

of data format: text, integer numbers, float numbers, processed/rendered images, videos, three-dimensional environments, synthesized sounds, etc (NARDO; PASTORE; BRIAND, 2015).

Outputs can be given in a wide range of formats, depending on the systems' business rule. Essentially, systems' outputs can be produced under two different ways: (i) instantly or (ii) separated by a given period of time due to pre-processing or, even, due to the necessity of a certain amount of data. Find below few examples of contemporary systems and their correspondent outputs:

- Text processing systems: in these systems, users interact with the application through a keyboard (input device), and the correspondent alphanumeric characters are instantly displayed in an electronic text (output);
- Speech to text systems: these software solutions are able to instantly convert spoken words (input) into written texts (output) through a process that is called speech recognition. Using a microphone device to capture human voices, after some pre-processing steps, the system matches the input with words, composing a written text as output;
- *Quick Response* (QR) code readers: in which a device reader (e.g. cameras) identifies a QR data (input) and produces a correspondent number as output; and
- Fingerprint scanner: these systems work through fingerprint sensors that consists of devices that capture a digital representation of a fingerprint (input). After applying some feature extraction algorithms, the system is able to produce an output indication if the fingerprint belongs to a person pre-registered in a database, or not.

The examples mentioned above are just a few contemporary IPOs. Due to the evolution of hardware and software systems, software/hardware designers are often proposing new IPO devices. To do so, designers Hardware features such as microphone, cameras, infrared, Bluetooth, smart chips and others are regularly used. Next, we present running examples of systems complex-output systems.

Regardless the IPO process and the systems application, systems' outputs are extremely important to V&V and testing activities of SE methods. Systems' output should be systematically evaluated by an automated method previously elaborated to check potential inconsistencies. However, sometimes the output domains is not friendly to automated implementation, configuring the complex-output domains.

Complex-output systems consist of software systems whose outputs' format are uncommon and unfriendly for automated software engineering activities. Further, complex-output systems are those whose outputs are provided in a non-usual data format. Voices, sounds, videos, graphics arrays, images, three-dimensional objects, statistical sampling, and multidimensional

data, are some examples of complex outputs that could be resultant from different types of software systems.

Complex-output systems are consequence of the advances of the programming languages and data structures that enabled developers to create more exquisite and modern applications, including the output data produced by those applications. [Shu, Bai e Podgurski \(2011\)](#) highlight several applications that trend to be intimately associated to complex outputs: computer games ([CRAVEN; FABRICATORE, 2016](#)), drawing tools ([BACH et al., 2017](#); [KWON et al., 2017](#); [VADERNA; DEJANOVIC; MILOSAVLJEVIC, 2016](#)), health information ([CASTRO et al., 2016](#); [LIKE et al., 2015](#)), interactive simulations ([SALMI; MAGREZ; ZIYYAT, 2014](#); [AL-BADRI, 2015](#); [SMITH et al., 2013](#)), TTS systems ([GUPTA; BANVILLE; FALK, 2016](#); [SURO; ONO, 2016](#)), bio-informatic programs ([CHEONG; LI; GOH, 2011](#)), GUI-based applications ([KOWALCZYK; MEMON, 2015](#)), much of the *Computer-aided diagnosis* (CAD) applications ([MA; ZHANG, 2016](#); [LIN; HUANG; CHENG, 2016](#)), and statistical systems ([DAHIYA, 2016](#); [PHAN et al., 2016](#)).

2.4.1 Testing challenges

Several testing challenges are intrinsic to complex-output systems. Among these challenges, we highlight three of them: (i) automatic re-execution of test cases; (ii) testing report generation; and (iii) automatic comparison among actual and expected outputs. Our research is associated with the third challenge that is the comparison among actual and expected outputs, since the this comparison depends on how friendly the SUT's output data are.

Testing automation has to do with checking SUT's inputs and outputs, regarding the SUT's specification. Then, we believe the SUT's output is the central point of testing automation activities. One of the most suitable definitions of software testing relies on the dynamism of the testing activities, highlighting that software testing is designed to exercise an SUT to assure the accordance of its outputs against some given inputs ([BERTOLINO, 2007a](#)). Hence, one can notice that this definition directly connects software testing with SUT's outputs. Then, from this definition, one can infer that software testing consists of systematic activities of feeding an SUT with inputs to check the correction of the current outputs.

Most of the contemporary complex-output systems produce output data whose correctness could be checked by a human through his censorial function such as hearing and vision. For instance, it is a trivial task for humans check whether a synthesized voice is correct or not. However, in a Software Engineering scenario, regarding testing issues, the complexity of the system's output is intimately associated with the possibility of implementing automated testing activities.

A challenge on testing complex-output systems relies on the application of testing techniques and criteria in separated components or specific units. Then, once software testing is

a dynamic process in which a tester exercises the SUT to check its correctness ([BERTOLINO, 2007a](#)), one can mention that complex-output systems shatter the literal definition of testing activities, leading testers to perform manual testing activities. Complex-output systems limits the application of automated testing activities, requiring the design of alternative and domain-specific testing strategies.

2.5 Final remarks and overview

This chapter has presented the most important concepts and background information for the complete understanding of the research. Software testing, test oracles, the oracle problem, and conceptual aspects on complex-output systems were widely presented. Next, we present a classical example of complex-output systems – TTS applications. Once TTSs are the central point of validation of this research, the next chapter presents general aspects of TTS systems as well as some of their particularities, i.e testing aspects and quality evaluation methods.

TTS - A COMPLEX-OUTPUT SYSTEM

3.1 Overview

Nowadays, there are several complex-output systems. Most of them are being used and poorly tested due to the lack of automated and systematic testing strategies. Automating generic testing activities to all of the complex-output systems is a remaining problem. This dissertation aims to contribute with this problem through a deep analysis on testing TTS systems through the support of automated test oracles. This chapter presents two relevant concepts: technical aspects on TTS applications (Sec 3.2) and a mapping study about testing aspects and quality evaluation of synthesized speech.

3.2 TTS: a complex-output system

TTS systems play an essential role in our daily life, being useful in net-banking solutions, travel direction orientation, automation of news and social media reading, automation of book reading, and assisting the interaction between blind people and computer-based technologies (TAYLOR, 2009). TTS-based applications are becoming pervasive in contemporary software applications, being useful web-applications or in desktop programs, in smart-phones, mobile and embedded systems, and so on. This section presents some conceptual and technical aspects of TTS systems.

3.2.1 *Concepts and definitions*

TTS systems implement a process to convert written text into synthesized human voice that must correspond to the natural reading of that text (TAYLOR, 2009). A TTS solution receives, as input, some written text in a given natural language and generates an audio output, corresponding to a sound of that speech (UNGUREAN; BURILEANU, 2011). Technically, a

TTS goals to product artificial voice through some computational process to read texts in natural language (TAYLOR, 2009). A TTS output can be represented by a *MPEG-1/2 Audio Layer 3* (MP3) or a *WAV Waveform Audio File Format*.

According to Taylor (2009), the process that enables a computer to read natural-language texts has three main macro-phases: (i) reading aspects; (ii) speaking strategies; and (iii) algorithms associated to methods that enables the association of written words and audio signals. The literature generically presents this field as “speech synthesis”, which means the generic creation of speech through computational structures. Speech synthesis systems need for an input written following the *American Standard Code for Information Interchange* (ASCII) pattern. Particular aspects of intonation, pauses and others are introduced through the punctuation. So, the correct punctuation of the input text is essential for the accuracy of the speech synthesis process.

Figure 10 presents a generic structure of components of TTS systems. Most of the components presented in the figure are mandatory to contemporary TTS systems.

Figure 10 – TTS’s generic structure

Source: Adapted from Taylor (2009).

There are four main modules for performing essential tasks for text-to-speech conversion. The *pre-processing* module (1) has the function of transforming the input text into word and punctuation symbols. This step deal with generic numbers, ordinal numbers, abbreviations, times and duration, dates, monetary amounts, telephone numbers, and units of measures. The *spelling-phonetic* conversion module (2) determines allophones (basic phonetic units) that compose each sentence using rules of transformation of sequences of letters into phonetic units. After that, a *synthesis* module (3) is responsible for generating the speech signal based on the list of phonetic units produced in previous modules. In this step, the synthesizer selects appropriate acoustic units in its database, processing and inserting rhythm, intonation to obtain speech. Finally, the *encoding* module (4) converts the generated output to the appropriate format.

Interpreting the operations of each module, it is possible to state that TTSs receive as input a sequence of ASCII characters of any size. To make proper processing, some algorithm that segment input into separate sentences is necessary. Therefore, the input data is divided into

a series of atoms, which preserve the white-space, punctuation, and so on. Often, atoms can represent individual words, numbers, dates, or phonemes. After this stage of processing, it is necessary to make semiotic classifications of each atom, seeking to identify symbols¹ and signs². Then, if there are languages that are not natural, it is necessary to decode the entries so that they become natural (text-to-text transformation). Problems of ambiguity must be identified and remedied by an appropriate algorithm. Finally, algorithms determine in an automated way the rhythm (intonation) and pronunciation of the sound to be synthesized before generating it.

Similarly to Taylor (2009), a study conducted by Alias *et al.* (2008) defines the creation of a TTS system as a manifold and complex task that involves several steps: (i) text analysis modules (normalization, pre-processor, morphological, contextual, etc); (ii) linguistic analysis (phasing, intonation, duration); and, finally, (iii) speech synthesis module (prosody generator and wave form generator, etc) (Alias *et al.*, 2008). In consonance to Alias *et al.* (2008), Figure 11 presents a specific structure of a TTS system implemented by Gerazov, Shutinoski e Arsov (2008).

Figure 11 – Generic structure of TTS systems

Source: Adapted from Gerazov, Shutinoski e Arsov (2008).

According to Figure 11, TTS systems need for a huge module called “Text Analysis Module”. This module performs three main tasks: (i) text normalization (text to text analysis); (ii) a mapping between words and pre-defined sounds; and (iii) prosody production that creates patterns of rhythm and sound. The second module consists of an “Internal Phonemic Representation”

¹ Symbols: a visible representative element that by analogy replaces an object

² Sign: elements that serve to represent “things in our brain”.

that represents an alphabet schema with phonetic notation. Finally, the third module – “Speech Synthesis Module” – is responsible to conclude the speech synthesis through concatenation, modification, and unit fetching of the signals. In this last module, characteristics of pitch, rhythm, intonation, and others are introduced through specific algorithms.

3.2.1.1 Speech signal processing

To the full comprehension of TTS systems, it is necessary to study concepts associated to techniques and strategies of speech processing and sound. The definition of sound is associated to a vibration that propagates through the air (through molecules) transferring the vibration forward in frequencies that can be picked up by receivers (ears) (MCLOUGHLIN, 2009; RABINER; SCHAFER, 1978). Otherwise, audio is a sound signal that has a broad range of frequencies (broadband signal) such as an audio signal of various musical instruments, animal sound signal, ambient noise, sound signal that combines musical instruments, voice, among other sounds and /or noises. Through comparisons between audio and voice, one can state that voiced signals have a smaller spectrum of frequencies (narrowband signal), being produced by the human speech generator system or even artificially synthesized. Thus, voice, or speech, consists of a subset of sounds that can be generated by the human being for their communication (SCHMANDT, 1994).

Digitized speech signals can be represented and visualized mainly in two different ways: (1) in the time domain; and (2) in the frequency domain. Tools such as the Fourier transform and convolution algorithms can be used to derive between the signal fields. Through those resources, it is possible to represent a voice signal scanned in the shape of a wave. Further, speech signals can have a graphical representation of positives and negatives values over time. This is because the human voice generated from the mouth causes the air pressure to be temporarily higher or lower than ambient air pressure.

Pronunciation and intonation problems impact visually in the digitized speech signal. Figure 12a and Figure 12b are running examples on how pronunciation and intonation aspects affect the graphical representation of speech signals. They present graphical signals of the Brazilian Portuguese words “acrobata” and “acróbata”. In this sense, the same word have different signal due to graphic accentuation.

Figure 13 presents two speech signals of the same sentence with different intonations. Figure 13a represents the signal from the sentence ‘Esta é sua casa.’ (in Brazilian Portuguese). Then, differences in the signal identity are notable when it is compared to the same sentence with interrogative intonation: “Esta é sua casa?” (Figura 13b).

A researcher or a practitioner can use algorithms for processing speech signal with several objectives. Among these objectives, we highlight: (i) analysis; (ii) synthesis; (iii) compressing; (iv) coding; (v) pattern recognition; (vi) speech recognition; (vii) pathology recognition; (viii) digital storage; and (ix) helping disabled people (JEKOSCH, 2005).

(a) Speech signal of “acróbata”

(b) Speech signal of “acrobata”

Figure 12 – Speech signal of the same word with different accents in the frequency-domain

Source: Adapted from [Mello \(2013\)](#).

Regarding this bunch of applications, the digital audio processing area has several resources that are well known in literature: digital filters, wavelets, fractals, neural networks, *Hidden Markov Model* (HMM), *Gaussian Mixture Models* (GMM), and others ([RABINER; SCHAFER, 1978](#); [SALIVAHANAN; ALAN, 2000](#)).

3.2.2 Quality limitation

The wide usage of TTS systems is severely limited due to their low quality. Different problems lead users to avoid using some TTS systems. In general, these problems are associated to the lack of reliability in the voices produced by TTS applications. Discontinuities, click sounds, contextual effects in wave concatenation methods are the most problematic aspects of TTS applications. Such factors could make the final voice so unnatural that some users consider using TTS systems as an annoying and wearing experience. In addition to that, once TTS's systems go through a manifold process, several other quality problems may occur.

The major problems of TTS systems are: (i) text-processing (normalization) problems

(a) Sentence with affirmative intonation

(b) Sentence with interrogative intonation

Figure 13 – Signal of the same sentence with different intonation

Source: Adapted from [Mello \(2013\)](#).

(e.g, dates, abbreviations, numbers, etc.); *(ii)* prosody problems (e.g, duration, intonation, stress); and *(iii)* pronunciation (e.g, homographs or similar-pronunciation words). According to [Taylor \(2009\)](#), there are several critical points on creating effective synthesized speech:

- Normalization: the identification and classification of symbols, signs, and icons in written texts is a recurrent problem for TTSs;
- Naturalness: the creation of natural speech, like real-human voices, is a challenge for practitioners;
- Affective and argumentative voice generation; and
- Speech adapted to the appropriate way that listeners need for in certain situations.

Despite the quality problems, researchers have been dedicating efforts on producing TTS applications since 1950s and this area is still attracting software companies and researchers. The significant time elapsed from 1950s endorses the importance of the accuracy of testing strategies

for TTSs applications. One can mention that TTS systems are still an immature research area due to the distance of SE methods and the development processes employed in TTS systems. Despite the subjectiveness and negligence of the automation of their testing activities, complex-output systems are still important in the daily human life. However, the process of making artificial human voices has several problems that makes the audio output looks unnatural or even impossible to understand.

Under the users' point-of-view, contextual effects and discontinuities are some of the most relevant quality issues in TTS systems (TAYLOR, 2009). These quality issues, generally, affects low-level synthesis in TTS applications. Regarding the developers point-of-view, female and child voices are the most challengeable to reproduce due to the uncommon pitch. For instance, female voices have a pitch twice as high as male voices and three time as high as children voices.

Automated strategies to check the quality of speech is a non-trivial task. Speech quality is a multidimensional term, making its automation process a domain-specific task. Thus, depending on the application, the evaluation method must be chosen carefully to achieve desired results. Next, we present a study mapping conducted to identify and categorize quality evaluation and testing strategies to TTS systems and speech synthesizer strategies.

3.3 Testing strategies and quality evaluation methods for TTS systems: a mapping study

Aiming at ensuring the originality of our study as well as investigating testing strategies and quality evaluation methods for TTS systems, we have conducted a Systematic Mapping on testing strategies and quality evaluation methods for TTS systems.

An Systematic Mapping (SM) is a repeatable and auditable document including the analysis over a pool of studies about a research topic (KITCHENHAM *et al.*, 2007). SMs are potential starting points for *Systematic Literature Reviews* (SLR) about some specific topic. Our SM addresses to identify clusters and research gaps on testing strategies or quality evaluation methods for TTS systems and synthesized speech.

3.3.1 Motivation

The main objective of the SM conducted in this study is to identify primary studies that report investigations about processes or strategies of verification, validation, testing or quality evaluation applied in TTS systems or synthesized speech. In this context, the MS has the following objectives:

- to identify techniques and strategies for testing activities in TTS systems;

- to identify clusters and research gaps on the quality and testing of TTS systems;
- to categorize test strategies for TTS systems;
- to evaluate and identify the most common problems in TTS system's outputs; and
- to identify cost factors and human efforts related to TTS systems.

3.3.2 Research Protocol

To ensure the replicability of our SM, all of the decisions and strategies we took during the conduct of the research are documented and justified through a Technical Report (OLIVEIRA; DELAMARO; NUNES, 2013). This technical report mainly presents the research protocol followed by us when conducting this research. It is available online for consults and analysis³. Some of the data contained in the report are: justification for the elaboration of search strings, selection of indexed databases, search strategies, inclusion criteria, exclusion criteria, quality criteria, etc. Then, the report presents all of the planning, conducting, and analysis phases. To support this dissertation, this section shows only the most important aspects of the SM.

Three Research Questions (RQ) guide our SM:

Question 1: Which testing activities have been applied in TTS systems?

Question 2: Which features of synthesized speech are considered in testing activities?

Question 3: Which empirical studies are conducted in testing activities to TTS systems?

The assessment of the testing techniques and quality evaluation methods for TTS systems, in the context of this study, aims at supporting the test oracle strategy proposed and validated through this dissertation. However, the term “test oracle” consists of a very particular word from the testing area. So, we did not include them in any RQ, avoiding including them in our search strings.

Figure 14 presents the generic search string of the SM.

(“Text-to-speech” OR “tts” OR “text to speech” OR “letter-to-phone” OR “letter-to-sound”)
AND
(“testing” OR “test”) AND (“technique” OR “strategy” OR “method”)

Figure 14 – Search string of the SM

Source: Elaborated by the author.

Figure 15 presents a flow that summarizes the way we conducted our SM, from the database definition until the final selection of papers. Six indexed online databases of scientific

³ see: <[www.labes.icmc.usp.br/\\$\sim\\$rpapes/dissertation.html](http://www.labes.icmc.usp.br/$\sim$rpapes/dissertation.html)>

papers were selected. Then, we adjust the generic search string (Figure 14) to each database. After the exclusion of repeated studies, unavailable studies, and the application of some filters, a pool of 379 studies was analyzed. Therefore, after reading summaries and applying our selection criteria (specified in the protocol), a pool of 141 pre-selected studies was obtained. Finally, after reading all of these studies, 115 studies were selected to synthesize the mapping. In the context of this research, the term study corresponds to any work or written article that presents some unpublished contribution, be it a primary or secondary study.

Figure 15 – *Workflow* followed in the mapping study

Source: Elaborated by the author.

3.3.3 Systematic Mapping Results

The general results and synthesis of the SM include quantitative data and information extracted from the complete reading of the studies. We identified several studies that seek to elicit characteristics and techniques for testing strategies and quality measurements of synthesized speech from TTS systems. Due to space limitations, in this section, we present only numbers and general analysis of the SM. Further details about the final pool of paper as well as the complete list of included studies are described in the technical report.

The bar plot presented in Figure 16 reports the selected studies, regarding each online database.

Based on the analysis of the studies selected, we defined answers to the pre-defined RQs. Then, following we present the answers to each RQ of our MS based on quantitative data.

Question 01: Which testing activities have been applied in TTS systems?

Figure 16 – Studies selection for each database

Source: Elaborated by the author.

The extraction of information from the studies allowed the identification of different methods and techniques with the same objective: to measure, access, and evaluate the quality of the synthesized speech produced by TTSs. Most the quality assessment activities identified in our study were routines, procedures, or strategies to empirically examine one or more aspect of synthesized speech perception to quantify these aspects (JEKOSCH, 2005). From this analysis, we can state that much of the test oracle strategy reported by the studies, even if not mentioned directly, take into account several factors related to signal quality. Concerning automation, most of the test oracles are human-based and ad-hoc.

In accordance the pool of studies we selected, when outputs in audio format are considered, methods to gauge the quality of the systems must take into account parameters that quantify and translate into values the perception of the user against any sound signal (VISWANATHAN; VISWANATHAN, 2005; ZHOU *et al.*, 2008). Several studies refer to the term quality evaluation of the outputs produced (instead of testing/verification/validation). Those studies, often refer to quality assessments mentioning some criteria of perception and understanding of the sentences generated by the systems. In some studies the perception criteria addresses one or more criteria firstly defined by Osgood (1952), namely:

- *Objectivity*: the output can be understood, interpreted, and passed on successfully by listeners;
- *Reliability*: the results do not change when the listener is changed. Then, any human being who is not aware of the particularities of the systems under evaluation;

- *Validity*: all of the parameters of the method are unique and related to perception issues;
- *Sensitivity*: the distinctions afforded by the method are as adequate as a listener analyzing the signs;
- *Comparability*: the method should be applicable in several contexts of perception and should allow comparisons between groups and conditions; and
- *Utility*: the information provided by the method is useful in the context for which the system was developed.

By associating the above criteria to the context of this dissertation, they give a preliminary view of the characteristics that should be explored when judging the adequacy of a TTs output. Thus, part of the research methodology can be supported by the SM. Regarding tools and features used to measure the quality of synthesized voices, two models of approaches stand out: (i) perceptual tests with humans or, therefore, (ii) measurements with physical (instrumental) equipment. The first approach is called “subjective” test (using listeners) and the second is called “objective” (using instruments) (SPAIS, 2005).

Subjective tests usually are performed with subjects, volunteers, or potential system users, and they are responsible for judging the quality of the output signals (TURK; SCHRODER, 2008). The subject listens to the SUT, filling a score card about several different aspects of the sound. These aspects are, generally, characteristics of perception (such as the characteristics defined by Osgood (1952)) or the similarity. Obviously, such as any process that depends on the intervention, interpretation, and decision-making of a human figure, these quality assessment methods are expensive, time-consuming, and poorly systematized (NORRENBROCK *et al.*, 2012; KREKOVIC; PRENNER, 2010).

On the other hand, some studies mention the usage of physical instruments (signal analysis and pattern recognition, specific statistical metrics, duration analysis, etc.)⁴ to measure the quality of synthesized speech. The studies that contribute these physical instruments present resources of very specific domains. Therefore, it is not trivial to define a clear and a well-established categorization of quality measures. In technical terms, these models used to measure quality can be implemented based on parameters that specify quality elements of the signal being evaluated. Also, such models may consider the signal being analyzed to process it and thus score and quantify its quality (VISWANATHAN; VISWANATHAN, 2005).

Most of the subjective evaluations use pre-defined patterns and protocols. An expressive number of studies have used a quality measurement called *Mean Opinion Score* (MOS). MOS is a wide used quality evaluation method for judging the quality of voices, regarding pre-defined score card. MOS is a standard for most of the practitioners, being the quality pattern recommended by *International Telegraph Union* (ITU) ITU (1996). A final MOS score corresponds to the average

⁴ other instruments are quoted in RQ 2

of the individual points of all of the subjects that evaluated a TTS under test. An example of MOS can settle that a given TTS should be judged following a scale: *excellent* (5 points), *good* (4 points), *fair* (3 points), *poor* (2 points) e *useless* (1 point) (VISWANATHAN; VISWANATHAN, 2005).

Figure 17 presents the main TTS quality evaluation methods identified by our SM. Besides MOS, one can highlight: *Diagnostic Rhyme Test* (DRT), which consists of the assessment of specific words and their capacity to be understood; and *Modified Rhyme Test* (MRT) that consists of the DRT extended for more words under different aspects. Besides the methods, several studies just classified their approach as formal or informal analysis, giving some subjectivity to the readers.

MOS was exploited by 71 studies. DRT was used in 37 studies. Professionals have used informal tests on 61 occasions. MRT was mentioned in 18 studies. Formal interpretations were used in 17 studies. Finally, 16 studies mentioned more than one quality assessment strategy.

Figure 17 – Methods to evaluate the quality of synthesized speech

Source: Elaborated by the author.

Question 02: Which features of synthesized speech are considered in testing activities?

Based on the answer formulated to RQ1, in accordance to what was revealed by the SM, the quality evaluation methods aim at reveals the following features:

- Fundamental frequencies (f0): the lowest and the highest frequencies that compose a harmonic series;
- Signal energy: signal general analysis, through specific algorithms or statistical data;
- Formant frequencies: the acoustic identity analysis of phonemes;
- Duration and pauses: the time of the sentence in milliseconds and the number of silences in the signal; and

- Speaker characteristics: basic identification and categorization of speaker speech such as normal, nasalized, infantile speech, whispered, elongated, animated, etc.

Question 03: Which empirical studies are conducted in testing activities to TTS systems?

Regarding of the studies included in the SM, we noticed a representative number of research goals. Some of the objectives we identified were algorithm proposals, performance analysis, and evaluation methods. In this context, we analyzed our pool of studies, selecting all of the studies addressing to answer RQ3. The following researches were identified: solution proposal (1), validation (2), research evaluation (3), experience report (4), philosophical studies (5), and others (6).

Solution proposals are studies that proposes new solution to well know problems with an argumentative line or example of application. *Research validation* is the category of studies of new techniques supported by experiments or case studies. *Research evaluation* is the group of studies that ultimately evaluates a method. *Experience report* is the category of study in which data are extracted from personal experience. *Philosophical studies* are those in which the authors present arguments to new paradigms. Finally, studies with different focuses were categorized as *others*.

Figure 18a presents the categories distribution performed in the SM. After selecting the final pool of studies, the research nature of the studies was distributed as follows: 21 research evaluation, six philosophical studies, ten experience report, 45 research validation, 30 solution proposals, and three other different classifications.

(a) Studies selection by their research nature

(b) Studies selection by their contribution

Figure 18 – Studies categorization by research nature and contributions

Source: Elaborated by the author.

In addition to the research nature analysis (Figure 18a) and aiming at support answering RQ3, we categorized each study by its contribution. Thus, each study was defined as being: *tool-paper* (21) (study focused on tools or systems modules and their applicability), *metric* (17) (studies that focus on metrics to evaluate existing techniques), *algorithm* (10) (studies that presents algorithms to different contexts), *techniques* (32) (studies that describes new techniques or improves existing techniques), *processes* (11) (studies new processes), *challenges* (9) (studies that aim at collaborating with classical challenges), *experiments* (7) (studies that conduct a experiment using a pre-existing resource) and, finally, *others* (8) (studies that do not fit in previous categories).

We created a catalog of the challenges related to regarding implementation, usage, and quality assessment of TTS and speech synthesis systems. Regarding studies that explicitly mentioned challenges, we highlight: quality issues were mentioned 15 times, seven studies referred to the process of converting graphemes to phonemes; prosody challenges were referred to in 11 studies, as well as 11 studies cited the unit selection process and finally, 13 studies mentioned particular challenges. Figure 19 present an overview of the challenges mentioned in the final selection of studies.

Figure 19 – Challenges revealed by the SM

Source: Elaborated by the author.

We categorized all of the future work mentioned by the studies, when it is referred to by authors. In this context, the following information was raised: nine authors mentioned other experiments in more complex scenarios, 43 studies mentioned improvements in the *unit selection* module of their approach, 17 studies indicated *prosody* increases and, finally, 14 mentioned particular future research. Figure 20 presents a bar plot that represents the main future work identified by the SM.

Figure 20 – Future work trend revealed by the SM

Source: Elaborated by the author.

3.3.4 Statistical data and demographic information

Through some descriptive statistics and demographic analysis, this section aims at presenting generic information about the state-of-the-art on the quality evaluation of synthesized speech.

Regarding the publication venues, we noticed more conference papers (91 studies) than journal/periodic papers (24 studies). Among the scientific conferences with the most occurrence of studies, we highlight *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* with 23 studies (Figure 21a). Regarding journal/periodic venues, we highlight *IEEE Signal Processing* with nine papers included in the final pool of studies (Figure 21b). Figure 21 brings a graphical representation with the most common conferences and journals identified through the SM.

Finally, regarding the year of publication, by means of the Figure 22, one can notice a two-year distribution of the number of studies since 2000. The figure reveals a constant growth in the number of papers treating quality issues in synthesized speech. For instance, in 2008 and 2009 we can notice a significant growth in the number of studies published about quality (18 studies).

On conducting this MS, many industry applications were identified. Hence, some of the studies selected were from real-world companies, instead of academic environments. Figure 23a presents a graphical representation of the proportion of academic studies (80.8%) and real-world industry cases (19.2%) included in our pool of studies.

Figure 21 – Studies selection by venue

Source: Elaborated by the author.

Figure 22 – Studies selected by year of publication

Source: Elaborated by the author.

3.3.5 General findings

This SM brought several findings to our research. Considering technical aspects of the implementation of speech synthesis and TTS applications, we can define that there are three basic models of systems: articulatory, formants, and concatenative methods (SAK; GUNGOR; SAFKAN, 2006). Articular models are complex because they are based on the human articu-

Figure 23 – Quantitative information about the SM

Source: Elaborated by the author.

latory system (vocal cords, vocal tract, etc.) (SAK; GUNGOR; SAFKAN, 2006; TURKMEN; KARSLIGIL, 2009). Formant-based voice synthesizers employ a series of rules to generate speech synthesis using phonemes formants, in particular, pre-defined frequencies for each phoneme (BARKHODA *et al.*, 2009).

Finally, in concatenative methods, which currently are the most common, Units (syllables, phonemes, etc.) of any language are stored in a database and, thus, the speech of words and sentences can be generated employing the concatenation of different units with different times (KIRKPATRICK; O'BRIEN; SCAIFE, 2006; CHAPPELL; HANSEN, 2002). Figura 23b presents a classification of all TTS system identified in our SM. We highlight the high number of concatenative TTS, representing 67.8% out of all studies with TTS.f concatenative TTS, representing 67.8% out of all studies with TTS.

Finally, regarding the language of the synthesizers, we highlight the high number of English-based studies (51), followed by Mandarin (22 studies). Portuguese was considered by only six studies, revealing the lack of TTS studies for that language. We considered six a low-expression number since Brazil has an active software industry, equally to the academic environment.

3.4 Final remarks and overview

This chapter has presented particular aspects of TTS systems and their details of implementation. A Mapping Study provided a bird-eye view over testing activities and quality evaluation methods to TTS systems. Next is presented a the test oracle approach proposed in this study to address to the challenge on defining automated testing activities to TTS systems.

THE FEATURE-BASED TEST ORACLE APPROACH

4.1 Overview

This chapter presents a novel test oracle paradigm to address to challenges of testing complex-output systems – the feature-based test oracles. According to the literature, software testing activities and complex-output systems are subjects that do not have any straight association. However, the importance of complex-output systems leads them to require systematic and automated testing activities. This chapter presents: (i) Details of a novel test oracle approach to complex-output systems (Sec. 4.2); (ii) An illustration of when and how to implement feature-based test oracles through a running example (Sec. 4.2.4); (iii) Details of a software framework that was extended to support the automation of testing environments for feature-based test oracles (Sec. 4.3); and (iv) Some final remarks revisiting the exploration of feature-based test oracles to complex-output systems.

4.2 Feature-based test oracles – designing automated test oracle to complex-output systems

This dissertation purposes and evaluates an approach called feature-based test oracles to address the implementation of automated testing activities of complex-output systems. This approach is based on the concept of exploring values of features extracted from a reliable output indicated by the tester. The values of these extracted features are used in our approach as a source of information to evaluate SUT's outputs. Then, when dealing with the absence of an expected output that attends to all of the SUT's outputs, testers can use some reliable output (handcrafted or collated from an alternative resource) to automate testing activities through test oracles. As soon as a reference to reliable features is defined, feature-based test oracles should provide

resources to the definition of test oracles based on the difference between the SUT's features and the same features collected from the reference output. Then, typically, the differences among SUT's outputs and the reference can be calculated by a similarity function. A distance of 0.0 means the perfect similarity and the difference between two objects is proportional to the value of the distance. Higher distance values mean several differences between the objects under comparison. These similarity functions (e.g. Euclidean distance), also called distance functions, can mathematically measure the difference between two objects through a vector of similarity values. Finally, the feature-based test oracles proposed through this dissertation explores feature extraction and object comparisons to automate test oracles.

Through these concepts, feature-based test oracles can be adjusted under three aspects: (i) the group of relevant features to a given test; (ii) the similarity function to measure the distance between SUT's outputs and the pre-defined references; and (iii) the maximum acceptable value of distance between the reference and the SUT's outputs. Then, to provide a final verdict (pass/fail), a feature-based test oracle requires a threshold value that represents the maximum of the value of difference among SUT's outputs and the references, considering their feature values.

Feature-based test oracles are recommended in the following cases: (i) when the SUT's outputs are complex and there is no standard method to provide a proper comparison of the SUT's outputs with some similar object; and (ii) when it's hard to define an expected output to the SUT's output. However, there are some well-succeed outputs from previous and similar implementation.

The concept of feature-based test oracles was adapted from CBIR concepts and previous attempts of automating test oracles for systems with graphical outputs (DELAMARO; NUNES; OLIVEIRA, 2013). To better demonstrate the idea of feature-based test oracles, Section 4.2.1 presents the concept of CBIR.

4.2.1 CBIR – Content-Based Image Retrieval

As an standardized definition, CBIR is any technology which helps retrieve digital image files using its visual content (DATTA *et al.*, 2008). Further, CBIR systems explore computer vision strategies to recover images from a set (or database) of images using a query image. Instead of using keywords, description or tags, CBIR systems analyze the contents of the query image and recover the most similar images in a database. In this sense, CBIR systems' goal is to locate, in an image database, the most similar images to a query image, according to one or more provided criteria (BUENO *et al.*, 2002). Then, CBIR systems allow recovering a finite set of images similar to a reference image. It is not the goal of a CBIR system to indicate if an image is identical to its query image, but rather identify its most similar images. Verifying whether images are the same could easily be done by using a pixel by pixel comparison.

In the CBIR context, the similarity criteria are obtained by extracting and comparing

characteristics from the images such as color, texture, edges detection, particular patterns, shape, etc. *Feature extraction* is the method used to obtain some information or particular data from the image. Extractors are procedures that can extract particular characteristics from the object under analysis. Then, CBIR uses a set of extracted features which creates a feature vector that will be considered in its retrieval. These extracted features represent the image at the time of the search (PAQUERAULT *et al.*, 2004). However, the feature vector is not sufficient to determine the recovery result. It is necessary to measure how similar two feature vectors are, using some sort of distance measure, called a *similarity function*. A distance/similarity function is an algorithm able to compare two arrays of characteristics, returning a non-negative value. The lower the value returned, the greater the similarity between the model image and the image produced by the SUT.

CBIR systems can be successfully implemented using different architectures. There are five steps that are used in this type of system: (i) acquisition, (ii) pre-processing, (iii) feature extraction, (iv) indexation, and (v) retrieval. Only two specific steps are used in this work to adapt CBIR concepts to provide a dynamic way to create test oracles – feature extraction and image comparison. Image comparisons are performed using the features extracted and the similarity function, which are algorithms that result in numeric values that represent the distance between the images.

4.2.2 Requirements and Definitions

Four elements are required to make the feature-based test oracle approach feasible:

1. Feature extractors: necessary to quantify a given extractable feature in value;
2. similarity functions: needed to measure the distance between a given reference and SUT's outputs;
3. Threshold definition: a value to define how acceptable are the differences among SUT's outputs and references; and
4. References: a reference from which features should be extracted and used as a source of information of testing activities. Typical examples of references are: handcrafted outputs, previous implementations of the SUT, SUT's outputs from similar applications, etc.

Once the four elements aforementioned are available, a tester can design testing activities from the perspective of the feature-based test oracle approach. Therefore, a test environment can be adjusted to allow testers to perform the following tasks:

- Selecting/adjusting a group of features that are important to be considered in a particular test purpose;

- Selecting/adjusting a similarity function to be used to measure the distance (difference) between the features of the SUT's outputs and a given reference;
- Indicating a reference or a group of references from which features should be extracted to serve as source of information to the test oracles;
- Indicating an SUT's outputs or a group of SUT's outputs; adjusting a threshold to be considered in the test as the tolerance limit of the difference between SUT's outputs and references; and
- Loading a given test, providing individual feedback of each test execution ("pass"/"fail").

Testing environments to feature-based test oracles can be implemented through testing frameworks or even with the support of traditional testing APIs to a particular programming language. Section 4.3 presents details of an extension of a pre-existent testing framework that was performed to make a pre-existent testing framework meets feature-based test oracle approaches. This framework (O-FIm/CO) was then widely explored in this dissertation's experiments.

4.2.3 Classification and particularities of feature-based test oracles

In comparison to other test oracle modalities provided in Section 2.3.4, feature-based test oracles have a particular characteristic – they explore features extracted from the SUT's outputs as a source of information to perform testing activities. Most of the test oracle methods are designed to produce some expected output. On the other hand, feature-based test oracles figure the distance of SUT's outputs and pre-defined references through a group of relevant features. For instance, machine-learning based test oracles use some *Artificial Intelligence* (AI) strategy to predict SUT's outputs. Equally, model-based test oracles use formal methods and formal specifications to create a resource (e.g a Finite-State Machine) from which SUT's outputs can be figured. Finally, metamorphic-based test oracle approaches depend on the definition of metamorphic relations to indicate the correctness of SUT's outputs. Then, in this context, the most valuable quality of feature-based test oracles is the usage of features extracted from actual SUT's outputs to produce a verdict.

The concept of complex-output system gives feature-based test oracles to a unique and particular test oracle approach in some cases. Feature-based test oracles are applicable in some complex-output domains that most of the other test oracle approaches cannot treat. For instance, considering the problem of testing activities to TTS systems, a discussion of this topic can be conducted through some questions (Q1 and Q2):

Q1 : Is it possible to implement an FSM or other formal structure to produce reliable synthesized speech from a text input? If yes, are these speeches reliable enough to be used in testing activities, playing the role of oracles?

Q2 : Is it possible to implement an AI structure to produce reliable synthesized speech from a text input? If yes, are these speeches reliable enough to be used in testing activities, playing the role of oracles?

One can claim that currently there are no evidence sufficient to answers the questions above. Maybe, in the future, the answers to both questions will be positive. In the current scenario, one can not affirm with certain that these implementations are possible and reliable enough to be explored in testing activities. However, a possible implementation of the solutions aforementioned should be much complex than the implementation of a TTS by itself, making their implementation costly to testing purposes. The same scenario is valid to other complex-output systems. Therefore, it is possible to affirm (with no experimental evidence) that feature-based test oracles have a favorable cost-benefit in comparison to most of the test oracle approaches to automate testing activities of complex-output systems.

Feature-based test oracles can be considered a sub-category of specification-based test oracles. This is due to two reasons: (i) the tester specifies the group of features that are relevant to this test, likewise the adequate parameters to them; (ii) feature-based test oracles evaluate some particular aspect in accordance with a specific testing purpose, then, in accordance to [Baresi e Young \(2001\)](#), these elements configure a specification-based approach.

4.2.4 When and how to use feature-based test oracles?

To better illustrate a situation in which the feature-based test oracle can be applicable, we will take an example regarding the test automation to medical systems.

Given a Program **P** that implements a medical system that produces synthetic three-dimensional objects, representing blood vessels. Then, it is necessary to design automated testing activities to evaluate **P**'s outputs, considering that its outputs are created in a data format that is read and, then, visually, reproduced by a particular medical system. The three-dimensional objects produced should be well evaluated since they are supposed to be explored in medical training of graduate students on providing a diagnosis to the following exams: (i) magnetic resonance angiography; and (ii) computed tomography angiography. **P** should afford high-quality outputs, providing a detailed analysis of blood vessels, enabling the adequate training of professionals.

In this introductory scenario, a tester can identify the applicability of feature-based test oracles providing answers to the following questions:

- Are the SUT's outputs complex? *Ans.*: Considering the aforementioned running example, the answer to this question is Yes. Three-dimensional blood vessels are sophisticated outputs, and the application of regular testing activities is useless to ensure the final quality of the outputs;

- Is there any reliable reference from which SUT's outputs can be explored? *Ans.:* In the context of blood vessels, there are some datasets of three-dimensional objects collected from exams in human subjects. These objects, generally are kindly provided by hospitals or research centers to be explored by researchers. Then, since these images were created from a reliable source of information and have a similar format to **P**'s outputs, one can claim that the answer to this question is "Yes";
- Are there any feature extractor or any library of feature extractors to the SUT's outputs and the references identified? *Ans.:* Regarding our running example, the answer is Yes. Bifurcation angles, vessels' width, total length, and medium diameter, are some of the well-known features that can be extracted from three-dimensional blood vessels. There are API's that implements these extractors, mitigating efforts on using them; and
- Are there any available framework or API that make it possible to explore feature-based test oracles for the purpose of testing **P**'s outputs? *Ans.:* Regarding the previous answers, one can mention that O-FIm/CO supports the automation of testing activities to this complex-output domain.

Through the questions and the answers exposed above testers certify that it is possible to implement feature-based test oracles to automate testing activities in **P**. Then, the tester should adapt the feature extractors to work properly in a given framework that supports feature-based test oracles (e.g., O-FIm/CO). After that, it will be possible to have an automated testing environment in which features are extracted from objects provided by a real-world medical exam and them compared to the same features extracted from **P**'s outputs. The tester can, then, define a starting point towards the automation of testing activities to a particular complex-output system.

Finally, from this running example, we claim that one of the most valuable contributions of this dissertation is the demonstration of the applicability of feature-based test oracles in several testing complex-output's domains. However, considering the impossibility of validating this approach in several domains and providing a narrow-scope to a single complex-output domain, we have validated feature-based test oracles with the purpose of automating testing activities to TTS systems.

Next, we present how we extended a testing framework previously implemented to support the feature-based test oracle approach.

4.3 The O-FIm/CO framework

O-FIm/CO is a testing framework extended in this study to support the feature-based test oracle approach. O-FIm/CO exploits a multidisciplinary strategy in which *Image Processing* (IP) techniques are adopted to provide a software testing resource for complex-output systems.

Besides technical and structural details of O-FIm/CO, this section presents overall concepts on using O-FIm/CO practice.

4.3.1 O-FIm/CO's structure

Figure 24 presents O-FIm/CO's structure. O-FIm/CO has an organization that matches different modules and resources in a common environment. These modules play fundamental rules to provide an oracle setting. These modules are: (1) “core”; (2) “parser”; (3) “wizard”; (4) “plug-ins”; (5) API; and (6) “oracle application”.

Figure 24 – O-FIm/CO General Structure.

Source: Elaborated by the author.

In accordance the feature-based test oracle approaches, one of the counterparts in using O-FIm/CO is the need for a “model/reference” from which it is possible to establish a baseline for comparing objects under test. For example, when it is desired to evaluate the appearance of a GUI in a particular *Look and Feel* (L&F), it requires an environment from which it is possible to establish a standard image (screenshot) for comparison during the test. This standard screenshot may represent, for example, a GUI in a default screen resolution and orientation. On the other hand, regarding TTS systems two empirical strategies are possible: (i) use two different systems and compare their outputs; and (ii) compare the output produced by a TTS system with the audio of the voice of a person who reproduces the text input. Figure 24 presents in details the generic structure of O-FIm/CO.

“Plug-ins” may represent features extractors for complex outputs or similarity functions. In the O-FIm/CO context, plug-ins are the main contribution from testers and they represent a reusable package of Java classes. In these plug-ins testers must implement algorithms able to quantify characteristics (feature extractors) or to figure out the difference between objects in accordance with their characteristics (similarity functions). In order to associate these plug-ins

and O-Flm/CO, testers must implement some specific Java Interfaces. Then, the requirement is that when developing an extractor, the tester creates a Java class that implements one of these Interfaces. There are different Interfaces for audio outputs, graphical outputs and similarity functions. After that, “Testers” are able to install/uninstall plug-ins in O-Flm/CO in two different ways: (1) using line commands that are interpreted by the O-Flm/CO “core”; (2) using a friendly “wizard”. Exploring *ClassLoaders*, O-Flm/CO loads plug-ins packages that able to act over an SUT (audio or graphical output) extracting relevant information to compose test oracles.

Once made the setup described above, testers may specify their test oracles using the O-Flm/CO “wizard”. To do so, feature extractors, extractor parameters and similarity functions must be specified. The final specification is saved in a text file that represents an “oracle description”. Figure 25 presents an example of oracle description including a similarity function, two extractors and their parameters. During a test execution, a “parser” analyzes oracle descriptions and set up the test. Finally, using an intuitive “API”, testers must write an “oracle application” that matches framework, oracle descriptions, SUT’s outputs and models. The oracle application is a Java program that analyzes a set of complex SUT’s outputs, extracting features specified by testers, returning if they are similar to a model or not, according to a threshold (precision).

```
-----
similarity Euclidean
extractor Colorcount { rectangle = [100 100 30 40] scale = 1.33 }
extractor Connectedcomponent { rectangle = [0 0 128 64] }
precision = 0.87
-----
```

Figure 25 – Generic Instance of an Oracle Description.

Source: Elaborated by the author.

4.3.2 O-Flm/CO’s API

Using O-Flm/CO API testers may implement useful oracle applications to support their decisions on testing activities. Among the API’s methods, we highlight four procedures as the most intuitive and useful: (1) `public void setDescOracle(String)`; (2) `public void setComplexData(String)`; (3) `public void setComplexSUT(String)`; and (4) `public double getOracleResult()`. Further, an oracle application can be associated to other automated test applications in wide testing scenarios.

After the framework settings, the tester has to write a program in order to automate the execution of test suites. To do so, the tester needs to implement an *oracle application*, in which the O-Flm/CO framework must be included as a library. In an oracle application, through a Java program, testers must define the source of information for their test (the baseline and standard outputs), the SUT, and the oracle descriptions. To make a testing environment possible,

Table 6 – Methods to be implemented by an O-Flm/CO plug-in

Method Signature	Specification
<code>public void setDescOracle(String)</code>	sets the path of an oracle description for a testing round
<code>public void setComplexData(String)</code>	specifies the path of a folder (or a file) containing the screenshots to be used as baseline for a test set
<code>public void setComplexSUT(String)</code>	defines the path of a folder (or a file) containing the screenshots to be used as source of information for the testing set
<code>public double getOracleResult()</code>	returns the result(s) of the application of the oracle regarding the testing set and baseline previously defined

O-Flm/CO has an API whose methods allows a straight communication with the low level of the framework. Through this API the tester will be able to load the oracle description, specify the SUT and specify the snapshot that is the baseline for a test. This application must to return a report specifying whether the tests passes or fails.

Among the API's methods, in Table 6 we highlight four procedures as the most intuitive and useful.

Further, through this API, an oracle application can be associated with other automated test applications in wide testing scenarios. Then, as a result of using O-Flm/CO, the tester has an environment in which it is possible to obtain a Java program that is able to repeatedly compare two images responding whether they are similar or not, according to visual features, similarity functions and a specific threshold.

4.3.3 Previous applications and O-Flm/CO in Practice

O-Flm/CO has been explored in three different contexts: CAD systems, web application, and GUI-based testing. In these three scenarios, visual and graphical aspects were considered to automate text oracles. In the context of this dissertation, O-Flm/CO will be extended to complex-output systems. However, regarding the previous application of the framework, we designed a generic workflow about the practical usage of the framework.

For instance, on using O-Flm/CO in the context of GUI-based systems required some effort investments from us. We spent an amount of time to set a favorable scenario to be explored several times. These investments required efforts on two different types of activities: (1) *testing strategy activities* on define what to test; and (2) *technical activities* towards on implementing the required plug-ins.

Strategic activities included a series of decisions-making on the testing strategy. Generally, these decisions are based on the knowledge of the tester about how the SUT must behavior under different situations. For instance, in the scope of this GUI-based testing, we have defined testing strategies into three steps:

1. *Defining basic visual requirements for the SUTs*: This step relies on assigning which are the minimum requirements to have a GUI-based system running like it is defined in the

SUT's specifications. So, we defined visual requirements that must be available for users all of the time such as, the font size, the visibility of some buttons, the proportional size of components, the color of specific widgets, etc;

2. *Establishing features to represent the selected visual requirements*: Defining features is inwardly associated with the two activities defined before. This activity required “know-how” on IP techniques and specific algorithms to determine a reliable set of extractable features to be represent the SUT in the moment of the test. In addition to specific features such as color, form, proportional size, depending on the SUT, we have included in these extractable features some visual important patterns; and
3. *Establishing in which conditions and environments the SUT must to be evaluated*: After defining basic visual requirements, it was necessary to establish in which circumstances the systems should be tested. Nominally, these circumstances regard on different L&Fs, OSs, screen sizes and resolutions, fonts size, etc.

In technical activities, all of the efforts required are associated to automate steps of the testing processes. In each one of the previous usage of O-FIm/CO, these activities were totally dependent of the set of decisions took in the testing strategy activities. We divided these activities into five minor specific activities:

1. *Finding or implementing feature extractors as O-FIm/CO plug-ins*: This process consisted of implementing feature extractors following the Java interfaces required by O-FIm/CO. However, O-FIm/CO has a catalog of plug-ins already implemented and validated. Once these plug-ins are reusable, we checked for useful feature extractors before start implementing the new ones;
2. *Selecting similarity functions as O-FIm/CO plug-ins*: In this activity we have looked for similarity functions that were able to measure properly the distance between SUTs' features and standard's features. Actually, we have selected similarity functions already implemented in previous contributions ([DELAMARO](#); [NUNES](#); [OLIVEIRA, 2013](#));
3. *Defining automated mechanisms for data collection*: This step, generically, depends on the STU's output domain and consists of implementing supporting technologies to collect data information for testing activities. Then, we have implemented mechanisms to automatically capture snapshots from the SUT under different conditions;
4. *Setting oracle descriptions using O-FIm/CO*: In this step of our experiment we have set different oracle description based on our previous knowledge about the SUTs' specifications. In this context, we went through each study case specification and, using O-FIm/CO, we set at least two different oracle descriptions to conduct our testing activities; and

5. *Implementing oracle applications*: Finally, in this activity, using O-Flm/CO as a Java library, we have set an environment to enable the automatic comparison among SUTs snapshots. These comparison were based on the specification defined in the oracle descriptions. After all, the oracle applications provided a report and a log reflecting the testing results.

Figure 26 represents graphically the workflow followed in previous usage of O-Flm/CO. In the figure, one can see all of the macro steps to use the framework successfully.

Figure 26 – Generic workflow associated with the usage of O-Flm/CO

Source: Elaborated by the author.

4.4 O-Flm/CO to TTS systems

Unlike images, an audio output is typically represented through a signal with frequencies that, generally, range of 20 to 20,000 Hz (*Hertz*). These frequencies are typically the representation of the sampling rate of the signal, which means that this represents the reduction of a continuous signal to a discrete signal.

Feature extractors for audio outputs could be designed under two different strategies: (i) short-term feature extraction; and (ii) mid-term feature extraction (MCENNIS D.; FUJINAGA., 2006; MCENNIS D.; DEPALLE, 2005). Short-term feature extractors split the signal into windows (frames) and compute a number of features for each frame. This process leads to a sequence of short-term feature vectors for the whole signal. Finally, all of the values from each frame can be quantified in one single value through an aggregation structure. Mid-term feature extractors are able to represent an audio signal by statistics extracted from short-term feature sequences. In this case, the extractors are represented by a number of statistics (e.g. mean and standard deviation) over each short-term feature sequence (MCENNIS D.; FUJINAGA., 2006).

Aiming at allowing the proper usage of O-FIm/CO in the context of automating test oracles for TTS applications, in this study we have adapted the framework to accept audio feature extractors and other generic features for complex-output purposes. Thus, some technical implementations were performed in the structure of the framework. After that, we have implemented a set of audio feature extractors. Following we explicit on how we performed the proper extensions in the framework to allow the test of complex-output systems.

The first technical/structural adaptation performed in the framework is the definition of a complete new Java interface for complex-output systems. Then, besides a specific Java interface to receive feature extractors to images, we have implemented a generic plug-in interface from which the tester can install his features. To do so, we have used the resources of the Java programming language know as multiple inheritance. Multiple inheritance enables that an object or class can inherit resources from more than one parent object/class. In the context of O-FIm/CO, we have implemented a generic abstract class for plug-ins called OFImPlugin, presented in Algorithm 6.

Algorithm 6: Abstract class for O-FIm/CO plug-ins

```

1 package br.oracle.pluginInterfaces;
2 public abstract class OFImPlugin {
3
4     public abstract String getName();
5     public abstract void setProperty(String propertyName, Object propertyValue);
6     public abstract Object getProperty(String propertyName);
7     public abstract Object[] getProperties();
8     public abstract String[] getPropertyNames();
9     public abstract double computeValue(String complexObject);
10 }

```

The abstract class OFImPlugin allows the application of multiple inheritance through the other types of plug-ins. Then, we have implemented new three Java interfaces to differentiate the O-FIm/CO plugin during, allowing their installation in the proper way. The three Java interface implemented are: (i) InterfaceSignalExtractor; (ii) InterfaceImageExtractor; and (iii) InterfaceSimilarityFunction. When specific methods are necessary in the particular context of the plug-in being developed, all of the required methods will be defined by the its correspondent Java interface. For instance, when implementing a Similarity Function, the tester should explore the methods specified by InterfaceSimilarityFunction. Algorithm 7 presents the source code of the Java interface for plug-ins that implement similarity functions

useful in O-FIm/CO.

Algorithm 7: Java interface for O-FIm/CO similarity function

```

1 package br.oracle.pluginInterfaces;
2 import java.util.Vector;
3 public interface InterfaceSimilarityFunction {
4
5     public String getSimilarityName();
6     public void addExtractor(OFImPlugin oFImPlugin);
7     public Vector<OFImPlugin> getExtractors();
8     public double computeSimilarity(double[] vectModel, double[] vectSUT);
9     public double[] getVectorSimilarity(String complexObject);
10 }
```

Then, when implementing some specific plug-ins, tester should make their java classes extends the abstract class `OFImPlugin` and implements some specific java Interface. Those settings are necessary to the proper administration of plug-ins by O-FIm/CO. In addition to that, through those implementations, once a plug-in is installed, it can be explored several times in testing environments, ensuring the flexibility offered by O-FIm/CO.

There are plug-ins with different complexities. Some of them will be more laborious than others, however this analysis is more related to the code of the extractor, than the association of this code with O-FIm/CO. For instance, when some tester has an API or, even, a catalog of feature extractors well implemented and validated, it is simple to adapt their code to work properly in the O-FIm/CO context. To illustrate a feature extractor implemented using O-FIm/CO, Algorithm 8 presents the source of a feature extractor that returns the duration of an audio sentence. It consists

of a real-world extractor that is often used in our experimental strategies.

Algorithm 8: Source code of an audio feature extractor for O-Flm/CO

```

1 package br.audioExtractor; import java.io.File;
2 import br.oracle.pluginInterfaces.InterfaceSignalExtractor;
3 import br.oracle.pluginInterfaces.OFImPlugin;
4
5 public class DurationExtractor extends OFImPlugin implements
    InterfaceSignalExtractor {
6 private double normalizer = 1.0;
7
8 public double computeValue(String pathToSound) {
9     File f = new File (pathToSound); Duration d1 = new Duration();
10    return d1.getDurationOfWavInSeconds(f)/getNormalizer();
11 }
12
13 private double getNormalizer() { return normalizer; }
14
15 private void setNormalizer(double normalizer) { this.normalizer = normalizer; }
16
17 @Override
18 public String getName() { return "Duration"; }
19
20 @Override
21 public Object[] getProperties() { return new Object[] {normalizer}; }
22
23 @Override
24 public Object getProperty(String propName) {
25     if (propName.toLowerCase().equals("normalizer"))
26     { return normalizer; }
27     else
28     { System.err.println("Invalid Argument ... "); return null;}
29 }
30
31 @Override
32 public String[] getPropertyNames() { return new String[] {"normalizer"}; }
33
34 @Override
35 public void setProperty(String propertyName, Object propertyValue) {
36     if (propertyName.toLowerCase().equals("normalizer"))
37     { try{
38         String doubleValue = (String) propertyValue;
39         setNormalizer(Double.parseDouble(doubleValue));
40     }
41     catch(Exception e)
42     {
43         System.out.println("Extractor problem..." + e); System.exit(0);
44     }
45     }else { System.err.println("Invalid Argument ... "); }
46 }
47 }

```

To evaluate effectiveness of the test oracles proposed in this study, we implemented several other feature extractors under different perspectives. For some of them, we use theoretical information and pre-defined formulas and values (e.g extractors of information from formant

and phoneme identification) and for others we have adapted algorithms in the context of our study (e.g Spectral centroid, Zero crossing, Spectral flux). At least 15 feature extractors explored in our experiments were adapted from source codes included in the JAudio¹ project. It consists of a software package to provide the sharing of knowledge through signal processing and audio feature extractors (MCENNIS D.; FUJINAGA., 2006; MCENNIS D.; DEPALLE, 2005). Besides generic audio extractors, JAudio provides generic features that are not totally related to speech processing. However, all of those features have high application in the context of our study.

Some of these feature extractors are: *Relative Difference function*, *Peak Based Spectral Smoothness*, *Partial based spectral centroid*, *Stringest frequency via Spectral centroid*, *Strongest Frequency via zero crossing*, *Strehngh of Stringest Beat*, *Beat sum*, *Stringest beat*, *Zero Crossing*, *Fraction of Low Energy windows*, *root mean square*, *Spectral variability*, *Compactness*, *Spectral flux*, *Spectral rolloff point Spectral Centroid*, and *Traditional Area Moments*.

The **appendix X (toDo)** presents details and code of all implemented feature extractors. Some of the main feature extractors used in this study are presented below. Details and descriptions of these features are presented as well. Generally, all of these feature extractors are applied in some sample (window) of the audio signal and, then, used to compute the final value of the extractors that is useful in O-FIm/CO's context.

Spectral centroid: this extractor is able to measure the center of mass of the power spectrum (Algorithm 9) of an audio signal. It is associated to sound brightness and timbres. It can be said that this feature is an indication of the amount of high-frequency content in an audio signal;

Algorithm 9: Spectral centroid extractor

```

1 public double[] extractFeature( double[] samples , double sampling_rate , double
    [[]] other_feature_values )
2     throws Exception {
3     double[] pow_spectrum = other_feature_values[0];
4     double total = 0.0;
5     double weighted_total = 0.0;
6     for (int bin = 0; bin < pow_spectrum.length; bin++)
7     {
8         weighted_total += bin * pow_spectrum[bin];
9         total += pow_spectrum[bin];
10    }
11    double[] result = new double[1];
12    if (total != 0.0){
13        result[0] = weighted_total / total;
14    }else{
15        result[0] = 0.0;
16    }
17    return result;
18 }
```

Compactness: A measure of the noisiness of a signal. Compactness sums over frequency

¹ see: <<http://jmir.sourceforge.net/jAudio.html>>

bins of an Fast Fourier Transform (FFT), providing an indication of the noisiness. This implementation, in particular, compares the components of a window's magnitude spectrum with the magnitude spectrum of its neighboring windows (Algorithm 10)

Algorithm 10: Compactness extractor

```

1 public double[] extractFeature(double[] samples, double sampling_rate,
2     double[][] other_feature_values) throws Exception {
3     double[] mag_spec = other_feature_values[0];
4     double compactness = 0.0;
5     for (int i = 1; i < mag_spec.length - 1; i++) {
6         if ((mag_spec[i - 1] > 0.0) && (mag_spec[i] > 0.0)
7             && (mag_spec[i + 1] > 0.0)) {
8             compactness += Math.abs(20.0 * Math.log(mag_spec[i]) - 20.0
9                 * (Math.log(mag_spec[i - 1])
10                    + Math.log(mag_spec[i]) + Math
11                        .log(mag_spec[i + 1])) / 3.0);
12         }
13     }
14     double[] result = new double[1];
15     result[0] = compactness;
16     return result;
17 }
```

Spectral flux: it consists of a one-dimension feature extractors that calculates the spectral Flux from a window of samples and the preceding window. It consists of a measurement of the difference between the current value of each magnitude spectrum bin in the current window regarding the same value of the previous one. The final calculation consists of the sum of the squares of the previous calculated differences. Algorithm 11 presents the implementation of the feature extractor.

Algorithm 11: Spectral flux

```

1 public double[] extractFeature( double[] samples,
2     double sampling_rate,
3     double[][] other_feature_values )
4     throws Exception
5 {
6     double[] this_magnitude_spectrum = other_feature_values[0];
7     double[] previous_magnitude_spectrum = other_feature_values[1];
8
9     double sum = 0.0;
10    for (int bin = 0; bin < this_magnitude_spectrum.length; bin++)
11    {
12        double difference = this_magnitude_spectrum[bin]
13            - previous_magnitude_spectrum[bin];
14        double differences_squared = difference * difference;
15        sum += differences_squared;
16    }
17
18    double[] result = new double[1];
19    result[0] = sum;
20    return result;
21 }
```

Zero Crossing: this feature represents how many times the waveform changed the sign. It is an useful metric to indicate noisiness and frequency of some event. It is considered a good measure of the pitch. Some definitions considering zero crossing a feature extractor for multiple application, representing the number of signal changing in an audio. Zero crossing implementations are very simple to perform. Algorithm 12 presents a possible implementation for a zero crossing algorithm.

Algorithm 12: Zero Crossing extractor

```

1  public double[] extractFeature( double[] samples, double sampling_rate,
                                double[][] other_feature_values ) throws Exception
2  {
3      long count = 0;
4      for (int samp = 0; samp < samples.length - 1; samp++)
5      {
6          if (samples[samp] > 0.0 && samples[samp + 1] < 0.0)
7              count++;
8          else if (samples[samp] < 0.0 && samples[samp + 1] > 0.0)
9              count++;
10         else if (samples[samp] == 0.0 && samples[samp + 1] != 0.0)
11             count++;
12     }
13     double[] result = new double[1];
14     result[0] = (double) count;
15     return result;
16 }
```

Traditional Area Method of Moments: this algorithm is a feature extractor that implements the statistical methods proposed by Fujinaga (1997). consists of an 2D method that reports the number of global chances through the signal. *Traditional Area Method of Moments* (TDA) is applicable in several contexts of audio processing and it seems very effective in experiments regarding the context of this study. TDA is a multidimensional extractor and it returns, generally, ten values that represents the signal. Algorithm 13 presents a code that implements a TDA

solution that we implemented in the O-Flm/CO context.

Algorithm 13: Traditional Area Moment extractor

```

1 public double[] extractFeature(double[] samples, double sampling_rate, double[][]
    other_feature_values) throws Exception {
2     double[] ret = new double[10];
3     double sum = 0.0;
4     for (int i = 0; i < other_feature_values.length; ++i) {
5         for (int j = 0; j < other_feature_values[i].length; ++j) {
6             sum += other_feature_values[i][j];
7         }
8     }
9     if(sum==0.0){ java.util.Arrays.fill(ret,0.0); return ret; }
10
11    for (int i = 0; i < other_feature_values.length; ++i) {
12        for (int j = 0; j < other_feature_values[i].length; ++j) {
13            double tmp = other_feature_values[i][j] / sum;
14            x += tmp * i; y += tmp * j;
15            x2 += tmp * i * i; xy += tmp * i * j;
16            y2 += tmp * j * j; x3 += tmp * i * i * i;
17            x2y += tmp * i * i * j; xy2 += tmp * i * j * j;
18            y3 += tmp * j * j * j;
19        }
20    }
21    ret[0] = sum; ret[1] = x; ret[2] = y; ret[3] = x2 - x * x; ret[4] = xy - x *
    y;
22    ret[5] = y2 - y * y; ret[6] = 2 * Math.pow(x, 3.0) - 3 * x * x2 + x3;
23    ret[7] = 2 * x * xy - y * x2 + x2 * y; ret[8] = 2 * y * xy - x * y2 + y2 * x;
24    ret[9] = 2 * Math.pow(y, 3.0) - 3 * y * y2 + y3;
25    return ret;
26 }

```

4.5 Final remarks and overview

in this dissertation: the feature-based test oracles. The oracle approach proposed addresses to automate testing activities of complex-output systems. Likewise, this section presents particularities of feature-based test oracles and their differences regarding other similar test oracle approaches. Further, this section presents a running example with a real scenario of medical systems to illustrate the applicability of feature-based test oracles from the testers' point-of-view. Finally, it is presented the extensions performed in a testing framework towards the adequate support to feature-based test oracles in practice.

EMPIRICAL EVALUATION

The last chapter proposed the use of feature based oracles to systems with complex output. Besides the proposition of this kind of oracle and its implementation as a framework for TTS systems, the present work also analyses aspects of its in several different scenarios. The contribution of this chapter is to establish which extractors are suitable to use in testing oracles and which quality aspects can be verified with them.

This section presents five empirical studies associating the automated test oracles proposed through this dissertation. The study in Section 5.1 investigates the effectiveness of using extractors of phonemes and formant to check the overall quality of singles words. The experiment in Section 5.2 presents a study to check if automated test oracles can rank sentences with slight differences, using as the reference an original sentence. Finally, the experiments in Section 5.3, 5.4, 5.5 and were conducted the check the applicability of test oracles in consonance with three real-world problems reported by practitioners: normalization problems (Sec. 5.3), the size of the audio databases (Sec. 5.4), and prosody inconsistencies (Sec. 5.5).

5.1 A pilot study on test oracles for TTS systems using feature extractors for formants and phonemes

This preliminary study aims at evaluating the feasibility of using methods of speech recognition to automate test oracles through O-FIm/CO. Then, this study's goals was to estimate time, cost, adverse events, and accuracy of exploring feature extractors implemented from techniques of speech recognition to perform useful automated test oracles. This pilot study consists of implementing feature extractors for formants and phonemes and validate them with the synthesized speech of real-world TTS systems.

Two well-known methods used in speech recognition are (i) formant frequencies (or formant analysis) and (ii) phoneme information. Then, the intention of this pilot study is to

investigate if information from formant frequencies and phonemes identification of speech signals of single words produced by real-world TTSs can be explored to automate test oracles. Next, we present technical details about the implementation of the feature extractors to be used in this pilot study.

5.1.1 Feature extractors implemented

Formant analysis/frequencies are different regions in the sound spectrum, or frequency bands, that form peaks of intensity (TAFNER, 1996). Formants are frequency bands that concentrate most of the acoustic energy of a sound signal. In speech signals, formants are the frequency components that characterize and provide meaning to human speech, allowing the hearing system to differentiate the vowels. The formants are commonly called **F1**, **F2**, **F3** ..., **FN**, sorted from the lowest to the highest frequency associated with them. According to Russo e Behlau (1993), there are several numbers of formants, but the first three are the most important. Formant analysis, depending on the language, is an essential process to identify vowels and phonemes.

Besides formant analysis, phoneme identification is a technique that is widely explored in processes of speech recognition. A phoneme is the smallest sound unit in a human speech (CALLOU; LEITE, 1995). In accordance to Bresolin (2003), phonemes are the audio unit that is capable of promoting significant differences among the speech sound, changing their meaning from word to word. One or more letters represent a phoneme in the graphical context. However, there is not always a correspondence between the number of phonemes and the number of letters in a given word. Also, the same letter may represent several phonemes. Each language has its set of phonemes. For instance, in Portuguese, there are 13 vowel phonemes, 19 consonants, and two semivowels.

We developed feature extractors using the concepts mentioned above. To do so, we have used some Java APIs to help to deal with *WAVEform Audio File Format* (WAVE) files. *Java Media Framework* (JMF)¹, *Java Sound*², and *Collection of Music Information Retrieval and Visualization Applications* (CoMIRVA)³ were some of the libraries used.

Then, two feature extractors originated from speech recognition techniques were implemented to support the conduction of this study. The subsections below present technical details about such extractors.

5.1.1.1 Vowel extractor

In Brazilian Portuguese, one can identify vowels by the frequency of their formants. To identify the vowels by means of the analysis of the formants, a researcher needs to know an acoustic correspondence between the vowel and the formant of the sound produced. This

¹ see: <<http://www.oracle.com/technetwork/java/index-jsp-140239.html>>

² see: <<http://www.oracle.com/technetwork/java/index-jsp-140234.html>>

³ see: <<http://www.cp.jku.at/comirva>>

correspondence can be identified through a scale of values relating the first two formants only, **F1** and **F2** (MIRANDA; MEIRELES, 2011). Figure 27 presents an hypothetical sound signal in the frequency domain with some F1, F2, and F3.

Figure 27 – Audio signal in the frequency domain and its formants

Source: Elaborated by the author.

In this scenario, we developed a feature extractor that can identify the existence of a given phoneme (identified through parameters) and the instant of its identification in milliseconds. Our feature extractor identifies the occurrence of the seven oral vowel phonemes of the Portuguese language, represented in Table 7. Thus, from a WAVE file containing a speech signal, the algorithm can return the number of occurrences of each vowel and the instant of time in which such vowels were identified in the file.

Table 7 – Brazilian Portuguese Vowels and respective examples

Brazilian Portuguese vowels and their frequencies	
Sound	Example
/a/	amor
/e/	extrator
/ɛ/	café
/i/	pilha
/o/	olho
/ɔ/	óculos
/u/	uva

Source: Elaborated by the author.

To identify the existence of a vowel, after identifying **F1** and **F2**, our algorithm follows the reference values presented by Miranda e Meireles (2011) to associate that values with some of the Brazilian Portuguese vowels. To do so, we use the Euclidean distance between **F1** and **F2** and the reference value of each vowel. Figure 28 presents a reference chart of each Brazilian Portuguese vowel.

Figure 28 – Chart of Brazilian Portuguese vowels

Source: Adapted from [Miranda e Meireles \(2011\)](#).

Algorithm 14 presents a code excerpt that corresponds to the main method of the feature extractor implemented. It is important to highlight that the complete feature extractor required several other implementation, such as an FFT (Fast Fourier Transform) algorithms to transform the signal obtained directly to the frequency domain, enabling to access values of the signal frequency at the amplitude peaks, identifying the formants.

To estimate the complexity of implementing this feature extractor, Table 8 shows some metrics of the final feature extractors implemented in format of a O-FIm/CO plug-in.

5.1.1.2 Phoneme extractor

The phoneme extractor implemented in this pilot study uses the *Mel-Frequency Cepstral Coefficients* (MFCC) approach to identify phonemes. MFCC is a form of represent the power spectrum of sound based in short-term extraction. The process of calculating MFCC is a widely used tool for speech recognition applications. Generally, a numerical vector containing relevant information from the signal spectrum for speech recognition ([COUTINHO; OLIVEIRA, 2012](#)). One of the theoretical foundations for the use of this method is that it approximates the behavior of the human auditory system when identifying the phonemes by means of a nonlinear scale.

In the context of this study, we implemented a phoneme extractor that is able to identify only the three main consonant segments of the Brazilian Portuguese: /k/ (10%), /t/ (9,4%) e /m/ (8,4%) ([VIARO; FILHO, 2007](#)). The algorithm splits the sound signal in time domain into short duration groups according to criteria such as time, average, maximum and minimum amplitude, and wave frequency. For the extraction of phonemes, the parameters for this division are configured to generate a larger number of groups. This is done because the duration of phonemes pronunciation is generally less than the length of the vowels, and also because of

⁴ see: <http://metrics.sourceforge.net/>

Algorithm 14: Main method of the Vowel extractor

```

1 public double computeValue(String complexObject) {
2     double totalQTD = 0;    double totalPOS = 0;    int    posicao    = 0;
3     String vogal;    String vogalAnterior = null;
4
5     File f = new File(complexObject);
6     WaveFile wave = new WaveFile(f);
7
8     wave.setConfigVogais(new Config("F","V", (String) this.getProperty(CONFIG)));
9
10    ExtratorGrupos g = new ExtratorGrupos(wave, "V");
11    int totalClusters = g.resultado();
12
13    for (int i=0; i<totalClusters; i++){
14        ExtratorFFT fft = new ExtratorFFT(wave, (int) (g.clusters[i][1] - g.
clusters[i][0]));
15
16        fft.resultado(wave.getSamplesData((int)g.clusters[i][0], (int)g.clusters[i][1]));
17
18        ExtratorFormVogal ev = new ExtratorFormVogal(fft);
19        posicao = (int) (g.clusters[i][2] / (Integer) this.getProperty(INTERVAL));
20
21        if (posicao < (Integer) this.getProperty(NTH)) continue;
22        vogal = ev.getVogal();
23
24        if (vogal.equals(this.getProperty(VOWEL)) && !vogalDuplicada(vogal,
vogalAnterior)){
25            totalQTD++;    totalPOS += totalQTD * posicao;
26        }
27        vogalAnterior = vogal;
28    }
29    wave.close();
30
31    if (this.getProperty(RETURN) == "P") return totalPOS;
32    else return totalQTD;
33 }

```

Table 8 – Metrics collect from the project of the extractors

Extractor	LOC	CC (mean)	# classes	# methods	# packages
Phoneme	2763	2.92	11	210	2
Vowel	2256	2.89	12	184	2

*Lines Of Code (LOC)**McCabe Cyclomatic Complexity (CC)*Numbers collect from the Eclipse plug-in called Metrics⁴

differences in the behavior of the algorithms against the different effects of vowel and consonant phonemes in the sound spectrum.

We applied the MFCC algorithm available in the CoMIRVA library to each generated group, and our algorithm compares the resulting vector with values determined by empirical analysis, performed by applying the algorithm to samples of the three phonemes of interest. The extractor also allows the configuration of parameters considering information such as age and gender of the speaker, as well as peculiarities of the signal synthesis system. These parameters are configured in an XML file.

Algorithm 15 presents the main method of the feature extractor implemented by the

phoneme extractor. Table 8, previously presented, shows the project metrics.

Algorithm 15: Main method of the Phoneme extractor

```

1 public double computeValue(String complexObject) {
2     double totalQTD = 0;    double totalPOS = 0;    int    posicao    = 0;
3     String fonemaAtual;    String fonemaAnterior = null;
4
5     File f = new File(complexObject);
6     WaveFile wave = new WaveFile(f);
7
8     wave.setConfigFonemas(new Config("F", "F", (String) this.getProperty(CONFIG))
9 );
10    ExtratorGrupos g = new ExtratorGrupos(wave, "F");
11    g.resultado();
12
13    for (int i=0;i<g.clusters.length;i++){
14
15        ExtratorMFCC mfcc = new ExtratorMFCC(wave, (int) (g.clusters[i][1] - g.
16 clusters[i][0]));
17        mfcc.resultado(wave.getSamplesData((int)g.clusters[i][0], (int)g.clusters[i
18 ][1]));
19
20        ExtratorFonema fonema = new ExtratorFonema(mfcc);
21
22        posicao = (int) (g.clusters[i][2] / (Integer) this.getProperty(INTERVAL));
23
24        if (posicao < (Integer) this.getProperty(NTH)) continue;
25
26        fonemaAtual = fonema.getFonema();
27
28        if (fonemaAtual.equals(this.getProperty(PHONEME)) && !fonemaDuplicado(
29 fonemaAtual, fonemaAnterior)){
30            totalQTD++;
31            totalPOS += totalQTD * posicao;
32        }
33        fonemaAnterior = fonemaAtual;
34    }
35    wave.close();
36    if (this.getProperty(RETURN) == "P") return totalPOS;
37    else
38        return totalQTD;
39 }

```

The next subsection details technical information about this study and information about the study's design.

5.1.2 Experimental Setup and Research Question

A single *Research Question* (RQ) guides our stud. The RQ and the metrics to support answering are:

- RQ1 What is the accuracy of feature extractors based on speech recognition techniques in the context of testing activities?

Metric I: Percentage of hits of the features. This metric and the number of false positives/negatives will be explored to support raising evidence to answer RQ1.

Aiming at answering RQ1, this empirical study consisted of checking the capacity of feature extractors designed from speech recognition techniques to be used in automated test oracles. Then, using a pool of 100 synthetic speech generated with two real-world TTS, we have evaluated the correctness and efficacy of the above mentioned extractors. We have chosen Brazilian Portuguese as the unique language of this pilot study because this is the native language of most of the researchers involved in this research.

Two real world TTS systems were used in this experiment. The first is the CPqD Texto-Fala (version 3.3)⁵ and Google Translate Text-to-Speech⁶. Both systems are widely used and their quality is well known by users. Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPqD) is the major Brazilian provider of telecommunications and solutions. TTS CPqD is a system able to synthesize texts written in Portuguese in speech signals close to human speech. Then, Portuguese is the idiom considered in this evaluation. CPqD Texto-Fala is a commercial TTS widely used in the automation of call centers, being essential in some sectors of society, such as the banking industry. Google Translate Text-to-Speech is widely used in web applications by difference services such as translations support, travel guidance, and others.

The experimental procedure designed in this study is simple. First, we generated the speech signal of the 100 words using the subject TTSs, creating a dataset that is available online for future replications⁷. The choice of words did not obey any restriction, having been obtained according to order of occurrence when we executed the TTSs with some random news. After that, through a human-based and ad-hoc analysis of each speech recorded, we have prepared a template with the expected answer to the implemented feature extractors. For instance, in the context of the vowel extractor, each word has a correspondent row into the table containing the number of occurrences of each vowel and a range of time in which each vowel was identified. Table 9 presents some numbers collected from our template.

Table 9 – Template of answers collected from the pre-defined dataset

Letter	Vowel Phoneme	# Occurrence	# Occurrence in different words
A	/a/	73	55
E	/e/	58	48
I	/i/	47	43
O	/O/	46	41
U	/U/	25	25
K	/k/	19	19
T	/t/	36	33
M	/m/	15	15

After collecting the dataset and adjusting our template with expected answers, both

⁵ see: <<http://www.cpqd.com.br/textofala>>

⁶ see: <<http://code.google.com/p/java-google-translate-text-to-speech>>

⁷ see: <<http://www.labes.icmc.usp.br/~rpaes/TTSExperiments.html>>

extractors went through a process of parameter calibration to work properly in the context of our TTS system using our data set including more than 100 words.

5.1.3 Experimental Results

We manually analyzed the results regarding the analysis with a pool of the 15 most common words. The most common words were defined by counting how many times each word were synthesized by CPqD TTS during the first steps of this experiment. Our test oracle provided 69.5% of hits and 30.5% of false alarms, in this particular case. Through an audio analysis we noticed that the different voices of the speakers may be the reason for the false alarms. Further, we have manually analyzed the results provided by each feature extractor regarding our whole data set with 100 words: Table 10 and Table 11 present a summary of errors and hits of the vowel extractor and the phoneme extractor, respectively.

Table 10 – Summary for the Vowel Feature Extractor.

Vowel	Number of occurrences					
	CPqD system			Google TTS		
	0	1	> 1	0	1	>1
A	71%	85%	57%	60%	68%	64%
E	65%	76%	60%	73%	61%	50%
I	89%	72%	50%	74%	59%	25%
O	75%	69%	40%	61%	72%	40%
U	68%	68%	100%	93%	40%	100%
hits	74%	74%	61%	72%	60%	56%
total	69.6%			62.6%		

Source: Elaborated by the author.

In Table 10 the number of the occurrences of the vowel in the words analyzed (0, 1 and > 1) is presented.

Table 11 – Summary for the Phoneme Feature Extractor.

Phoneme	Number of occurrences			
	CPqD system		Google TTS	
	0	1	0	1
K	74%	53%	81%	63%
T	85%	48%	68%	52%
M	73%	47%	52%	20%
hits	77%	49%	67%	45%
total	63.0%		56.0%	

Source: Elaborated by the author.

5.1.4 Experimental Analysis

The results obtained show, in the first analysis, similar data between the two subject TTS applications. The results were figured from minicious manual analysis in the dataset defined in this study. This result indicates that there is independence of the algorithm with respect to the TTS system. Through this analysis, we can conclude that the extractor works similarly for both systems, presenting success or failure in similar cases.

It is also possible to notice that the extractors tends to fails when there is more than one occurrence of the target event in the same word. This can be explained by the fact that, given the probability of error in identifying an occurrence individually, the probability of errors in identifying more than one occurrence of the same vowel/phoneme increases.

Finally, the general average of precision and accuracy reveals that the number of hits is not high, and then, using these feature extractors in testing environments is not recommended.

Once this study's hypothesis is that the feature extractors are reliable enough to perform well in automated test oracles, we executed binomial test with the data we obtained. In this case, in particular, the binomial test helps to check if the difference in the number of hits and errors of the extractors is due to coincident or if it reveals a systematic difference with some significance.

In this context, we define a null hypotheses (**H₀**) that states as follows:

H₀: “there is no difference between the results of our extractors in the probability of hitting the number of phonemes/vowels in a given speech signal”.

Then, **H₀**: errors = hits = 1/2;

In order to perform a proper binomal test, we established a necessity of 95% confidence. Then, it is decided that α should be less than 0.05 to reject **H₀**.

The results reported in Table 12 shows that the number of hits reached by the feature extractors are significant and in three occasions it is possible to reject H₀ trough the data obtained. Further, in one single case (Phoneme extractor with audio generated from Google TTS) the obtained data are not enough to draw conclusions.

Table 12 – Binomial test performed at a 95% confidence interval

Hits Rate	Vowel		Phoneme	
	CPqD 69%	CPqD 62%	Google TTS 63%	Google TTS 56%
P-Value Result	0.0001 Reject H ₀	0.0105 Reject H ₀	0.0060 Reject H ₀	0.1356 Accept H ₀
Meaning	High chances of observing more than 69 cases of success in 100 evaluations.	This is the chance of observing 62 or more successes in 100 evaluations.	This is the chance of observing 63 or more successes in 100 evaluations.	This is the chance of observing 56 or more successes in 100 evaluations

5.1.5 Threats to Validity

Find below the threats to the validity of this study:

- Conclusion: The main conclusion threat of this study is associated with two reasons: (i) low statistical power of the binomial test and (ii) low number of subject applications. We believe that it necessary more similar studies involving more real-world TTSs;
- Internal: An internal threat of this study is associated with the process of recording the speech. Thus, after recording each word, we needed to normalize the audio signals to obtain better results. We have used the software audacity to do so.;
- Construct: We did not find any relevant construct threat. However, one can mention that our subjective analysis to generate a template of answers can be defective and could be improved in future occasions;and
- External: Still regarding the template development, a possible external threat to this is fact that only one researcher defined the template. To mitigate this, another researcher has reviewed the template, contributing to improve this process.

5.1.6 Recommendations and Findings

In this sense, besides the statistical analysis, we consider the accuracy of the extractors lower than the necessary to be applied in testing activities, in particular to support the automation of test oracles. Using these extractors in a generic testing environment can brings two unwanted situations:

- High number of false results: when some artifact of a given testing environment is not working properly, it is common to observe a high number of false positives/negatives in the final results; and
- High trade-off on implementing test oracles: Due to necessity of calibration for each audio signal, these kind of feature extractors could be costly to be applied in real-world testing scenarios. Lack of flexibility and need for re-factoring in some code excerpts are some of the well-known problems associated to this kind of artifact.

Then, the overall finding of this study states that feature extractors based on speech recognition are too costly to be adapted in flexible testing environments. Due to this, in the following experiments we present empirical results using more generic audio feature extractors. The reason to this could be the fact that feature extractors to speech recognition are highly-dependent to parameters to enable their exploration in specific domains. Then, then this can demand a specific study to identify adequate parameters to improve the feature extractors effectiveness in the context of this study.

5.2 Towards on defining effective test oracles to TTS systems

The main goal of this study is to promote the empirical analysis of feature-based automated test oracles for the purpose of their evaluation with respect to effectiveness and efficiency from the point of view of the tester in the context of real-world TTS applications. The experiment addresses to raise evidences on creating text oracles able to automate the detection of real TTS problems, i.e. finding, automatically, TTS problems (intonation, pronunciation, prosody, text pre-processing, etc). Thus, our experiment at checking whether output-feature-based test oracles are able to differentiate TTS's output, working as a test oracle in testing activities.

It's important to highlight that this study is the first of a series of incremental studies aiming at check the effectiveness of feature-based test oracles for systems with complex outputs. Then, it aims at checking whether slight alterations in some sentences are considered by feature-based test oracles.

5.2.1 Research Questions

In order to guide our experiment, we have designed an empirical study towards raising evidences to answer the following RQs:

- RQ1 : Are CBIR-based test oracles applicable and reliable in the context of alleviating human efforts in testing TTS applications? - *Metrics*: Precision, recall;
- RQ2 : Is there any significant difference between test oracles considering the nature of their extractors (short-term and mid-term)? - *Metrics*: Precision, recall, statistics;
- RQ3 : Is there any significant difference between the usefulness of the test oracles depending of the language? - *Metrics*: Precision, recall, statistics;
- RQ4 : Is there any significant difference between the usefulness of test oracles depending on the TTS voice (male or female)? - *Metrics*: Precision, recall, statistics.

All of the metrics used to support answering our RQs are defined in Section 5.2.6.

5.2.2 Experimental Design

This' study's design addresses to support answering all of the RQs aforementioned. In the context of the applicability and reliability of the technique (**RQ1**), we use measurements such as, “*precision*” and “*recall*”. In pattern recognition and information retrieval (e.g. CBIR), precision (also called positive predictive value) is the fraction of retrieved instances that are

relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved.

Aiming at promoting a testing environment, we guide our study simulating a situation in which the CBIR-based test oracles should be able to differentiate mutated versions of an original sentence in a given level of relevance. To do so, starting from an original sentence, we intentionally created several “mutated” sentences that are supposed to make the original sentence increasingly different from the mutant versions. Find below an example of an original sentence and the mutants created from it:

Original: “Only Hard Choices for Parents Whose Children Flirt With Terror”

Mutation 01 (significant letter): “Only Had Choices for Parents Whose Children Flirt With Terror”

Mutation 02 (syllable): “Only Had Choices for rents Whose Children Flirt With Terror”

Mutation 03 (word): “Only Had for rents Whose Children Flirt With Terror”

Mutation 04 (word): “Only Had for rents Whose Children Flirt With”

Mutation 05 (Other sentence): “This sentence has nothing to do with the original sentence”

It is possible to observe that each mutant is applied in a given sentence, making it more unacceptable, regarding the original one. Then, the purpose of making those muted versions is to measure the sensibility of CBIR-based automated test oracles on identifying how different the mutants are, taking into account the features extracted from the original version of the sentence. We have measured precision and recall of each sentence, considering Mutants 1, 2, and 3 as the three first mutants to be identified as similar to the original. In order to make the results more reliable, we have used three sentences for each language.

Aiming at raising evidences to answer **RQ2**, we have defined three different test oracles. The first one (**O1**) was designed using only short-term audio feature extractors, the second one (**O2**) was designed with the same features of the first one, however all of the values were normalized using min-max values. Finally, the third one (**O3**) was designed using only mid-term audio feature extractors.

Regarding **RQ3**, five languages were selected to be part of the study: English, German, Portuguese, French, and Finnish. In addition to that, to address to **RQ4**, we have evaluated the TTS applications with male and females voices. This investigation is justified is due to the natural difference of frequencies that exists between child, female, and male voices.

The experiment was designed to follow the steps:

- Select different five languages: English, Portuguese, French, German, and Finnish;

- Select six different real-world TTS systems to compose the experiment;
- For each language, pick three different headlines from real-world news web sites;
- Perform five incremental mutations (changes) in the original headline, making them different from the original;
- record the audio output of the five different TTSs using different female and male voices; and
- use three CBIR-based test oracles strategies to differentiate the original and the muted version of each sentence, observing precision and recall of each oracle.

5.2.3 Subject TTSs

The study involves six real-worlds TTSs. Table 13 presents each TTS, the mode that we have collected the audio output, the sampling rate, and the frequency of the audio file. It is important to highlight that we put all of the audio files available in the format of a dataset, making it possible to use it in future empirical analysis⁸.

Table 13 – Subject TTS used in the study

TTS	Data Collec.	S. Rate	Freq.(KHz)	#Samp.
Google TTS	Native wav	16	16	30
IVONA	Audacity	16	44.1	60
FromTextToSpeech	Native wav	16	22.05	24
OddCast TTS	Audacity	16	44.1	108
ACapela TTS	Audacity	16	44.1	84
CWSpeak	Audacity	16	44.1	96

Source: Elaborated by the author.

5.2.4 Experimental procedure

In order to conduct the experiment, we have performed several technical tasks. After setting a Java testing application using the framework O-FIm/CO with the audio feature extractors, we have performed the following steps:

- Data collection: we exercised all of the TTS using all of the sentences selected and their respective mutants. Some of the TTS had a feature to save a “.wav” file representing the TTS output. However, when the subject TTS did not support the persistence of the file, we have used the free software audacity⁹ to record the sound and save so our test oracles can evaluate it;

⁸ see the dataset repository: <<https://goo.gl/J1D4zS>>

⁹ see: <<http://www.audacityteam.org/>>

- Data normalization: after having all of the audio files saved, we have performed a pre-processing task aiming at trimming the audio files, cutting the initial and final silences;
- oracle definitions: three oracles were defined O1 (using short-term extractors), O2 (using normalized short-term extractors), and O3 (using a mid-term statistical extractor);
- results collection: in addition to our test oracles, we instrumented our testing application so all of the features extracted and the data about the audio files were written in a sheet file;
- results preparation: all of the results of executions were stored in sheets files to be properly interpreted and analyzed towards answering our RQs.

5.2.5 Experiment conduction

Then we have started the process of running the tests using the artifacts developed to instrument the applications. We have created a Java program to run all of the CBIR-based test oracles over the outputs. Then, we stored all of the results in pre-defined spreadsheet. Once we have five sentences to be evaluated by our CBIR-based test oracle, we set our precision and recall rules following some strategies in accordance to three variable **A**, **B**, **D**, and the number of hits on ranking the sentences in accordance to the original sentence: **A**: sentences that are relevant and were retrieved considering the three first sentences retrieved (true positive); **B**: sentences that are relevant and were not retrieved considering the three first sentences retrieved (false negatives); **D**: sentences that are irrelevant and were not retrieved, considering the three first sentences (true negative); and Hits on ranking **H**: the number of sentences that were correctly ranked.

Then, our precision is calculated through $A/A + B$ and our recall is $H/A + B$. Table 14 summarizes all results collected for one TTS in particular – Google TTS calculated from all of the sentences recorded in Female-German voices. It is possible to observe that sometimes the precision is the same for all oracle strategies, however the recall can change significantly.

All of the results, their interpretation, and possible answers for our RQs are presented next.

5.2.6 Experimental Results and Discussion

In this section we interpret and discuss all of the results associated to the experiment detailed above.

Table 15 presents the overall results collected from the experiment. The table presents information of precision and recall associated to each test oracle and each sample of information evaluated in the study. One can notice the Precision ranged from 0.80 to 0.95, reaching an average of 0.88, which means a high score for retrieval purposes and an acceptable average for testing purposes. On the other hand, the recall metric ranged from 0.42 to 0.64 (average 0.55), which is

Table 14 – Data collected using Female-German voices in Google TTS

Oracle	Sent	A	B	D	Precision	RankHits	Recall
O1	s1	2	1	1	0.66	2	0.4
	s2	3	0	2	1	2	0.4
	s3	3	0	2	1	2	0.4
	tot.				0.88		0.40
Oracle	Sent	A	B	D	Precision	RankHits	Recall
O2	s1	2	1	1	0.66	2	0.4
	s2	3	0	2	1	3	0.6
	s3	3	0	2	1	3	0.6
	tot.				0.88		0.53
Oracle	Sent	A	B	D	Precision	RankHits	Recall
O3	s1	2	1	1	0.66	3	0.6
	s2	3	0	2	1	3	0.6
	s3	3	0	2	1	5	1.0
	tot.				0.88		0.73

Source: Elaborated by the author.

still good for retrieval purposes, but it is not acceptable for automated testing purposes. Then, we conclude that the answer to the **RQ1**, which addresses to the general usage of the strategy, is that the strategy is not mature enough to automated and deterministic testing purposes, however the automated test oracles might be acceptable to alleviate human efforts on checking the quality of TTS applications with previous references e.g, in regression testing activities.

Table 15 – Overall results considering all samples

Dat.	#	PO1	RO1	PO2	RO2	PO3	RO3	P	R
Pt-F	42	0.80	0.54	0.94	0.59	0.90	0.76	0.88	0.63
Pt-M	30	0.92	0.63	0.92	0.53	0.92	0.68	0.92	0.61
En-F	54	0.84	0.48	0.86	0.45	0.84	0.63	0.85	0.52
En-M	54	0.80	0.41	0.79	0.31	0.81	0.53	0.80	0.42
De-F	48	0.84	0.42	0.83	0.41	0.87	0.54	0.85	0.46
De-M	54	0.86	0.54	0.83	0.38	0.83	0.45	0.84	0.46
Fr-F	48	0.90	0.51	0.92	0.45	0.96	0.75	0.93	0.57
Fr-M	48	0.94	0.51	0.92	0.58	0.88	0.65	0.91	0.58
Fi-F	18	0.93	0.53	1.00	0.73	0.93	0.64	0.95	0.64
Fi-M	12	0.92	0.60	0.89	0.48	0.89	0.63	0.90	0.57
tot.	408	0.87	0.52	0.89	0.49	0.88	0.63	0.88	0.55

acronyms: **F**: Female - **M**: Male - **Pt**: Portuguese - **En**: English
- **De**: German - **Fr**: French - **Fi**: Finnish - **POx**: Precision of
oracle X - **ROx**: Recall of oracle X - **P**: Precision - **R**: Recall

Source: Elaborated by the author.

Regarding **RQ2** addressed to the nature of the test oracles (short-term versus mid-term), our experiment revealed that **Oracle 3**, composed by mid-term statistical feature extractors, performed slightly better than the other two test oracles. This could be analyzed regarding the high statistical power of the mid-term extractors. Figure 29 displays a visual information about the individual performance of each test oracle.

Figure 29 – Overall analysis of the test oracles

Source: Elaborated by the author.

Considering **RQ3** which addresses to check whether our test oracles are more suitable to some languages, our experiment revealed that French (precision of 0.92 and recall of 0.58) and Finnish (precision of 0.92 and recall of 0.60) were the most friendly languages to CBIR-based test oracles. English seemed to be the most unfriendly language to CBIR-based test oracles. Table 16 shows the precision/recall metric for each language and gender individually. We believe that some languages performed better because of the origin of the idioms and the nature of the languages. For example, Portuguese and French are both languages created from Indo-European Italic primitive languages, justifying their similarity in performance. On the other hand, Finnish is one language that the phonetic values are regular and they are pretty much similar to the written text, without much of tonal expressions.

The experiment shows that female voices are more suitable to be evaluated using our test oracles than TTS implemented through male voices (RQ4). This was expected due to the difference of frequencies between male and female voices. The speech of an adult male can have a fundamental frequency that ranges from 85 to 180 Hz. Similarly, an adult female from 165 to 255 Hz. This indicates that further studies are necessary to propose strategies to mitigate these

Table 16 – Precision/Recall by language

Lang.	Female		Male		Average	
	P	R	P	R	P	R
Pt	0.88	0.63	0.92	0.61	0.90	0.62
En	0.85	0.52	0.80	0.42	0.82	0.47
De	0.85	0.46	0.84	0.46	0.84	0.46
Fr	0.93	0.57	0.91	0.58	0.92	0.58
Fi	0.95	0.64	0.90	0.57	0.92	0.60
tot.	0.89	0.56	0.87	0.57	0.88	0.54
acronyms: Pt : Portuguese - En : English - De : German - Fr : French - Fi : Finnish - P : Precision - R : Recall						

Source: Elaborated by the author.

differences of performances of the automated test oracles.

These results raise some evidences about the need for flexible code independent testing strategies. Due to advances of the information systems, much of the contemporary applications produce complex outputs, making conventional testing approaches ineffective. CBIR-based test oracles follow a trend that might be adopted by several practitioners that need for specific-domain automated testing strategies.

5.2.7 Threats to validity

In this experiment, the threats to validity are considered over the four standard aspects: (i) internal validity; (ii) external validity; (iii) construct validity; and (iv) conclusion validity.

Internal validity: internal validity are influences that can affect the independent variable with respect to causality, without the researcher's knowledge. We avoided all of the internal threats through our research design strategy. However, an internal validity we identified is the different source of feature extractors algorithm. In other words, we have used feature extractors from different areas of the signal processing and they were originally designed for different purposes. This heterogeneity may causes some unpredictable side-effect in our results.

External validity: the external threats to the validity are associated to the capability of generalize the study to real-world industry problems. Regarding this issue, we identify one central threat: (i) interaction of setting and treatment: once we are making different artificial alterations on the original sentences we selected, one can mention that the test oracles could not address to real-world different sentences, being effective and useful only in simulation scenarios.

Construct validity: the construct validity are associated with the generalization of the results. Regarding this issue, the threats are twofold: (i) confounding constructs and levels of construct. The presence or the absence of prior knowledge in signal processing may affect the results regarding better or worse combinations of feature extractors; and (ii) restricted generality

across constructs. The different configurations of test oracles may affect positively the automated rank of some sentences in determined language, but unintentionally it can affect other rank configuration negatively.

Conclusion validity: this threats are associated with the ability to draw the correct conclusion about the effect most relevant threats regarding the conclusion validity are: (i) the reliability of the feature extractors implementations. Wrong implemented feature extractors may affect the results negatively or positively; (ii) the random heterogeneity of the TTS subjects. The TTSs we selected may not represent real scenarios of their usage.

5.2.8 Final remarks

The process of making artificial human voices has several problems that make the audio output looks unnatural or even impossible to understand. Generally, quality evaluation activities and testing activities for TTS systems are manual, ad-hoc, and unproductive. Implementing automated test oracles to support testing TTS systems is far from trivial. This is due to fact that TTS applications configure a complex-output system. This study proposes and evaluates an alternative to evaluate testing efforts of TTS systems through automated test oracles. Through experiments evolving six real-world TTSs the purposed automated CBIR-based test oracles seem to be a promising strategy to alleviate manual testing efforts. Empirically, the test oracles seem to be acceptable to supporting activities during the phase of test and validation of TTS application. The generic precision of the test oracles ranged from 0.95 to 0.80 depending on the language and gender of the TTS. Equally, the recall metric ranged from 0.42 to 0.64. We conclude that this automated test oracles can compose a step towards a collaboration between Software Engineering and practitioners that develop complex-output systems.

In comparison to our previous study, the feature extractors explored in this experiment, in particular, Traditional Area Method of Moments, Spectral Centroid, and Spectral Flux seem to perform better and in a more favorable scenario to be used in automated testing strategies. Next, we present study cases that show how feature-based test oracles deals with real-world sub-problems of TTSs such as normalization issues, different databases size, and differences of pitch, rhythm, and others sound aspects.

5.3 Automated text oracles to check normalization problems in TTS systems

Text pre-processing is the first step on the artificial production of artificial human voice. Text pre-processing activities are in charge of converting raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called text normalization, pre-processing, or tokenization. Text pre-processing represents complex task

in speech synthesis due to the fact that it deals with very specific language problems:

- Digits and numerals must be expanded into full words;
- Fractions and dates are also problematic;
- Expansion ordinal numbers have been found also problematic;
- Roman numerals may be also confused with some common abbreviations;
- Abbreviations;
- Special characters and symbols;
- Acronyms must be analyzed and expanded sometimes; and
- Numbers may also have some special forms of expression.

The test of pre-processing aspects in TTS systems are often a manual and unproductive process. This study aim at verifying the effectiveness of feature-based automated test oracle on detecting pre-processing problems in TTS systems. Given this scenario, this study aims at analyzing automated test oracles for the purpose of their evaluation with respect to effectiveness and reliability from the point of view of the tester

The definition of this experiment in terms of objects of study, purpose, perspective, focus, and context are:

- Object of study: This study aims at verifying the effectiveness of automated test oracles to identify problems in a particular aspect of TTS systems: the pre-processing. In this case, the objects of study are the test oracles;
- Purpose: The purpose of the study is to evaluate the effectiveness of the test oracles regarding their power on detecting preprocessing faults in TTS systems.
- Perspective: The perspective associated to the experiment is from the tester's point of view;
- Quality focus: The effect studied in the experiment is associated with the effectiveness (# faults) and reliability (# false/true positives and false/true negatives) of the test oracles.
- Context: The experiment consists of selecting five multilingual real-world TTS systems (subject programs) and generating different outputs from sentences. These sentences include several candidates that tend to make the TTS system crash. Using a model TTS (a TTS output normalized by hand), the experiment consists of implementing test oracles that are able to figure the distance (difference) between the model and the subject application.

5.3.1 Research Questions and Metrics

This study is guided by a single RQ:

- **RQ 1:** Are feature-based test oracles able to identify normalization issues in real-world TTSSs?

Once we are evaluating test oracles effectiveness, the metric that supports answering our RQ is the number of false positives and false negatives of the test oracles.

5.3.2 Experimental Design and subjects

Based on RQ 1, the experimental design of this study defines a scenario in which automated test oracles can evaluate the actual TTSSs' outputs in order to identify normalization inconsistencies.

Then, we have defined the following experimental procedure:

- Choosing five real-world TTS (in Brazilian Portuguese with a female reader). Section [5.3.3](#) presents all of the subject TTS explored in this study;
- Selecting 30 critical sentences to raise the chance of causing normalization problems in TTSSs' outputs;
- Collecting the “.wav” file generated by these 30 sentences as outputs in the five selected TTSSs. These are called actual sentences, since they were actual outputs produced by the TTSSs and their own normalizers;
- Normalizing the 30 critical sentences by hand. This process consists of write in full the meaning of all symbols, abbreviations, numerals, etc. of each sentence;
- Collecting the “.wav” file generated by these hand-normalized 30 sentences in the five selected TTSSs;
- Composing two feature-based test oracles with different configuration to compare the hand-normalized sentences and the actual sentences generated by the TTSSs;
- Performing a human-oracle evaluation comparing hand-normalized and the actual sentences, producing a verdict (pass or fail) depending on the content of the actual sentences; and
- Counting the number of false positives/negatives produced by the automated test oracles in comparison to the human-based analysis.

Table 17 presents all of the 30 sentences explored in this study. All of those sentences were collected from real-world headlines from Brazilian news websites. The table also presents the version of the sentence normalized by hand. Finally it is described the normalization critical aspect included in each sentence. Acronyms, percentage, numerals, currency values, country/state abbreviations, and fractions are some of the critical aspect identified. In order to avoid biases, we have randomly selected the headlines, however, one can notice that most of the sentences include more than one critical aspect. For example, Sentence #24 includes five different normalization issue, making it a good “test case”, once it exercises the TTS normalizer in five different rules.

Table 17 – Sentences synthesized and their correspondent hand-normalized versions

#	Sent.	Written sentence/Hand normalized version	Normalization Issue
1	Orig. Norm.	Previdência custará 20% do PIB em 2060. <i>previdência custará vinte por cento do pib em dois mil e sessenta</i>	percentage, acronym, and year.
2	Orig. Norm.	Segundo MPF, 13% dos quase 500 mil candidatos do país estão sob suspeita. <i>segundo eme pe efe, treze por cento dos quase quinhentos mil candidatos do país estão sob suspeita.</i>	acronym, percentage, and numerals.
3	Orig. Norm.	Líder do PSDB pede para PGR investigar nomeação de Lula. <i>líder do pe esse de be pede para pe ge erre investigar nomeação de lula.</i>	acronym (twice).
4	Orig. Norm.	CPI aponta fraude de R\$ 3 bilhões em fundos de pensão e 200 envolvidos. <i>ce pe i aponta frauda de três bilhões de reais em fundos de pensão e duzentos envolvidos.</i>	acronym, currency value in Reais, and numerals.
5	Orig. Norm.	Policial militar é baleado pela esposa durante briga no ES. <i>policial militar é baleado pela esposa durante briga no espírito santo.</i>	state abbreviation.
6	Orig. Norm.	Brasil perdeu 1,5 milhão de empregos em 2015. <i>brasil perdeu um milhão e meio de empregos em dois mil e quinze.</i>	numerals, and year.
7	Orig. Norm.	Festa do Cinema em maio com bilhetes a 2,5 € nas salas aderentes. <i>festa do cinema em maio com bilhetes a dois virgula cinco euros nas salas aderentes.</i>	currency value.
8	Orig. Norm.	ES precisa de 34.200 mesários voluntários para eleições de 2016. <i>espírito santo precisa de trinta e quatro mil mesários voluntários para eleições de dois mil e dezesseis.</i>	state abbreviation, numeral, and year.
9	Orig. Norm.	STF desbloqueia R\$ 2 bilhões da OAS. <i>esse te efe desbloqueia dois bilhões de reais da o a esse.</i>	acronym (twice), and currency value.
10	Orig. Norm.	MEC libera abertura de 2,3 mil vagas de medicina em 37 cidades. <i>mec libera abertura de duas mil e trezentas vagas de medicina em trinta e sete cidades.</i>	acronym, and numerals (twice).
11	Orig. Norm.	PB tem mais de 300 vagas em concursos públicos. <i>paraíba tem mais de trezentas vagas em concursos públicos.</i>	state abbreviation, and numerals.
12	Orig. Norm.	Assaltos crescem 31,54% em Pernambuco no primeiro trimestre. <i>assaltos crescem trinta e um virgula cinquenta e quatro por cento no primeiro trimestre.</i>	percentage.
13	Orig. Norm.	STF julga inconstitucional reajuste concebido a servidores pelo TJ-RJ. <i>esse te efe julga inconstitucional reajuste concebido a servidor pelo te jota rio de janeiro.</i>	acronym (twice), and state abbreviation.
14	Orig. Norm.	ONU condena uso de gás lacrimogêneo contra refugiados. <i>onu condena uso de gás lacrimogêneo contra refugiados.</i>	acronym.

Continued on next page

Table 17 – continued from previous page

#	Sent.	Written sentence/Hand normalized version	Normalization Issue
15	Orig. Norm.	30% dos eleitores brasileiros já contam com biometria. <i>trinta por cento dos eleitores brasileiros já contam com biometria.</i>	percentage.
16	Orig. Norm.	“Eleitor não precisa aguentar o governo”, diz ministro do STJ. <i>eleitor não precisa aguentar o governo, diz ministro do esse te jota.</i>	acronym.
17	Orig. Norm.	Trabalho infantil aumenta no Estado de São Paulo, diz IBGE. <i>trabalho infantil aumenta no Estado de São Paulo, diz i be ge é.</i>	acronym.
18	Orig. Norm.	62 municípios concentravam cerca de 50% do PIB em 2013, diz IBGE. <i>sessenta e dois municípios concentravam cerca de cinquenta por cento do pibi em dois mil e treze, diz i be ge é.</i>	numerals, percentage, acronym, and year.
19	Orig. Norm.	Desemprego cai em dezembro, mas renda tem 1º recuo desde 2005. <i>desemprego cai em dezembro, mas renda tem primeiro recuo desde dois mil e cinco.</i>	ordinal numeral, and year.
20	Orig. Norm.	AM registra queda de 1,5% na indústria em julho, mostra IBGE. <i>amazônas registra queda de um e meio por cento na insdústria em julho, mostra i be ge é.</i>	state abbreviation, percentage, and acronym.
21	Orig. Norm.	Após vaga na USP, filho de pedreiro sonha estudar em Harvard. <i>após vaga na usp, filho de pedreiro sonha em estudar em rarvard.</i>	acronym.
22	Orig. Norm.	Governo dos EUA gasta US\$ 1,4 milhão em aplicativo que demora apenas 10 minutos para ser feito. <i>governo dos estados unidos gasta um vírgula quatro milhão de dólares em aplicativo que demora apenas dez minutos para ser feito.</i>	country abbreviation, currency value, and numerals.
23	Orig. Norm.	1/4 dos contribuintes fez a entrega do IRPF em Sorocaba. <i>um quarto dos contribuintes fez a entrega do i erre pe efe em sorocaba.</i>	fraction, and acronym.
24	Orig. Norm.	PIB dos EUA sobe 1,4% na taxa anualizada do 4º tri de 2015 na leitura final. <i>pib dos estados unidos sobre um vírgula quatro por cento na taxa anualizada do quarto trim de dois mil e quinze na leitura final.</i>	acronym, country abbreviation, percentage, ordinal numeral, and year.
25	Orig. Norm.	CE vai receber 2 milhões de vacinas contra H1N1 nos postos de saúde. <i>ceará vai receber dois milhões de vacinas contra aga um ene um nos postos de saúde.</i>	state abbreviation, numerals, and acronym.
26	Orig. Norm.	Chove em Fortaleza e mais 58 municípios cearenses nas últimas 24 horas. <i>chove em fortaleza em mais cinquenta e oito municípios cearenses nas últimas vinte e quatro horas.</i>	numerals (twice).
27	Orig. Norm.	Perda de autonomia começou em 2015, dizem gestores de hospitais do CE. <i>perda de autonomia começou em dois mil e quinze, dizem gestores de hospitais do ceará.</i>	year, and state abbreviation.
28	Orig. Norm.	Governador do CE envia à AL lei que regulamenta promoções de militares. <i>governador do ceará envia a alagoas a lei que regulamenta promoções de militares.</i>	state abbreviation (twice)
29	Orig. Norm.	Caminhonete é recuperada a 3.5 mil KM do local do roubo. <i>caminhote é recuperada a três ponto cinco mil quilômetros do local do roubo.</i>	numerals, and abbreviation.
30	Orig. Norm.	Agronegócio respondeu por 1/3 da exportação brasileira. <i>agronegócio respondeu por um terço da exportação brasileira.</i>	fraction.

Source: Elaborated by the author.

We claim that this study was designed in consonance with practitioners from CPqD. Then, we planned the study to reproduce and follow as much as possible the real-world problems and issues faced by a software team that implements commercial TTS systems in Brazil. Besides the written sentences, the complete dataset of audio collect through this research is available online for future replication¹⁰.

Two different configuration of test oracles were defined:

- *Oracle 01*: This test oracle aims at verifying whether some regular feature extractors can work together to identify normalization issues during the speech processing. Then, this test oracle was configured with sixteen feature extractors:
 - Spectral Centroid;
 - Spectral Rolloff Point;
 - Spectral Flux;
 - Compactness;
 - Spectral Variability;
 - Root Mean Square;
 - Fraction Of Low Energy Windows;
 - Beat Sum;
 - Strength Of Strongest Beat;
 - Strongest Frequency Via Spectral Centroid;
 - Strongest Frequency Via FFT;
 - Partial Based Spectral Centroid;
 - Peak Based Spectral Smoothness; and
 - Relative Difference Function.

Technical details of each feature extractor explored in this study can be found in Appendix ???. Further, we used 0.2 as the overlap of windows and 64 as the frame size to be evaluated. The mean of results is used as the aggregator to compose the final value of the features. This combination of features is common in solutions of Content-Based Audio Retrieval (CBAR). The results of the feature extractors were normalized using the min-max strategy. Due to space limitation, some technical information were omitted in this manuscript, further information about the feature extractors and their specification can be find online¹¹.

¹⁰ see: <linkhere>

¹¹ see: <[www.labes.icmc.usp.br/\\$\sim\\$Rpaes/dissertation.html](http://www.labes.icmc.usp.br/$\sim$Rpaes/dissertation.html)>

- *Oracle 02*: This test oracle addresses to check if an effective statistical-based test oracle is able to identify normalization problems in TTS outputs. Then, this test oracle uses a single multidimensional feature extractor – *Traditional Area Method of Moments*. This feature extractor implements a 2D statistical methods of moments with ten dimensions, which is very useful in music retrieval solutions. Again, the min-max strategy was used to normalize the results.

As we defined in the first test oracle, we used 0.2 as the overlap of windows and 64 as the frame size.

The test oracles were implemented through O-FIm/CO and we added several functionalities in that framework to collect several information useful for our study, such as: value of each feature, and time to load the test oracles. All of these information were collected in an XML file to allow deeper analysis on the behavior our technique.

5.3.3 Subjects

Table 18 presents details of the subject TTSs explored in this study.

Table 18 – TTS subjects explored to check test oracles against normalization issues

TTS	Voices	Access/Link
CPqD Texto-Fala	Female	https://www.cpqd.com.br/textofala/
Ivona	Fem./Male	https://www.ivona.com/
ODDCast	Fem./Male	http://www.oddcast.com/home/demos/tts/
VocalWare Demo	Fem./Male	https://www.vocalware.com/index/demo
VozMe	Fem./Male	http://vozme.com/index.php

Source: Elaborated by the author.

5.3.4 Experimental Results and Discussion

For each subject TTS, two tables of results were generated regarding each test oracle configuration. For instance, Table 19 shows the detailed results of “Oracle 02” applied in the CPqD TTS. The results of all of the ten dimensions of the feature extractor is presented, as well as the Euclidean distance between the sentence normalized by hand and the actual TTS output. The number in the table represents the feature result of each dimension of the extractor Traditional Area Method of Moments.

Appendix A presents all of the the tables with results of all test oracles, including the analysis of all TTS subjects.

Table 19 – Detailed results of Oracle 2 applied in CPqD TTS

Sent	TDA01	TDA02	TDA03	TDA04	TDA05	TDA06	TDA07	TDA08	TDA09	TDA10	Euc. Dist.
Phr01-Norm	0,7895	0,0063	0,0077	0,0001	0,0001	0,0001	0,0000	0,0001	0,0001	0,0000	0,0197
Phr01-Real	0,7876	0,0189	0,0227	0,0004	0,0004	0,0005	0,0000	0,0004	0,0005	0,0000	
Phr02-Norm	0,8369	0,0357	0,0409	0,0013	0,0015	0,0017	0,0000	0,0015	0,0017	0,0001	0,0338
Phr02-Real	0,8264	0,0567	0,0647	0,0032	0,0037	0,0042	0,0002	0,0037	0,0042	0,0003	
Phr03-Norm	0,9343	0,0740	0,0828	0,0055	0,0061	0,0069	0,0004	0,0061	0,0068	0,0006	0,0281
Phr03-Real	0,9535	0,0888	0,0960	0,0079	0,0085	0,0092	0,0007	0,0085	0,0092	0,0009	
Phr04-Norm	0,9457	0,1058	0,1137	0,0112	0,0121	0,0129	0,0012	0,0119	0,0128	0,0015	0,0353
Phr04-Real	0,9668	0,1238	0,1331	0,0153	0,0165	0,0177	0,0019	0,0163	0,0176	0,0024	
Phr05-Norm	0,8961	0,1399	0,1496	0,0196	0,0209	0,0224	0,0027	0,0208	0,0222	0,0033	0,0385
Phr05-Real	0,9283	0,1534	0,1627	0,0235	0,0250	0,0265	0,0036	0,0248	0,0263	0,0043	
Phr06-Norm	0,9794	0,1660	0,1730	0,0276	0,0287	0,0299	0,0046	0,0286	0,0298	0,0052	0,1710
Phr06-Real	0,8095	0,1796	0,1835	0,0323	0,0329	0,0337	0,0058	0,0328	0,0335	0,0062	
Phr07-Norm	0,8813	0,1951	0,1983	0,0381	0,0387	0,0393	0,0074	0,0385	0,0392	0,0078	0,0625
Phr07-Real	0,8269	0,2114	0,2179	0,0447	0,0461	0,0475	0,0095	0,0459	0,0473	0,0104	
Phr08-Norm	0,8923	0,2313	0,2408	0,0535	0,0557	0,0580	0,0124	0,0556	0,0578	0,0140	0,0586
Phr08-Real	0,9329	0,2537	0,2647	0,0643	0,0672	0,0701	0,0163	0,0671	0,0700	0,0186	
Phr09-Norm	1,0000	0,2711	0,2839	0,0734	0,0769	0,0806	0,0199	0,0769	0,0805	0,0229	0,0881
Phr09-Real	0,9153	0,2835	0,2966	0,0804	0,0840	0,0879	0,0228	0,0840	0,0878	0,0261	
Phr10-Norm	0,9277	0,2983	0,3111	0,0890	0,0928	0,0968	0,0266	0,0927	0,0967	0,0301	0,0687
Phr10-Real	0,8717	0,3167	0,3315	0,1003	0,1049	0,1098	0,0318	0,1050	0,1098	0,0364	
Phr11-Norm	0,8803	0,3320	0,3487	0,1103	0,1157	0,1215	0,0366	0,1158	0,1215	0,0424	0,0265
Phr11-Real	0,8807	0,3441	0,3611	0,1183	0,1242	0,1303	0,0407	0,1243	0,1304	0,0470	
Phr12-Norm	0,8139	0,3597	0,3779	0,1294	0,1360	0,1429	0,0466	0,1361	0,1430	0,0540	0,0424
Phr12-Real	0,8153	0,3789	0,3958	0,1436	0,1500	0,1567	0,0544	0,1501	0,1568	0,0620	
Phr13-Norm	0,8507	0,3996	0,4173	0,1597	0,1667	0,1741	0,0638	0,1669	0,1742	0,0727	0,0982
Phr13-Real	0,9320	0,4205	0,4405	0,1769	0,1853	0,1941	0,0744	0,1854	0,1942	0,0855	
Phr14-Norm	0,9465	0,4366	0,4556	0,1904	0,1988	0,2075	0,0831	0,1989	0,2076	0,0945	0,0294
Phr14-Real	0,9459	0,4482	0,4664	0,2008	0,2089	0,2175	0,0900	0,2092	0,2177	0,1014	
Phr15-Norm	0,8382	0,4600	0,4779	0,2116	0,2198	0,2285	0,0974	0,2200	0,2286	0,1092	0,0686
Phr15-Real	0,7767	0,4728	0,4878	0,2234	0,2305	0,2379	0,1056	0,2306	0,2380	0,1160	
Phr16-Norm	0,8955	0,4868	0,4985	0,2369	0,2425	0,2484	0,1153	0,2426	0,2484	0,1238	0,0418
Phr16-Real	0,9050	0,5016	0,5122	0,2516	0,2569	0,2623	0,1263	0,2569	0,2623	0,1344	
Phr17-Norm	0,8597	0,5158	0,5257	0,2659	0,2710	0,2762	0,1371	0,2710	0,2762	0,1452	0,0600
Phr17-Real	0,9057	0,5290	0,5383	0,2798	0,2846	0,2896	0,1480	0,2846	0,2895	0,1558	
Phr18-Norm	0,8028	0,5467	0,5579	0,2989	0,3049	0,3112	0,1635	0,3049	0,3111	0,1736	0,0795
Phr18-Real	0,8218	0,5697	0,5829	0,3245	0,3320	0,3399	0,1849	0,3319	0,3396	0,1982	
Phr19-Norm	0,8890	0,5899	0,6006	0,3480	0,3542	0,3606	0,2054	0,3540	0,3603	0,2165	0,0498
Phr19-Real	0,8889	0,6066	0,6132	0,3681	0,3720	0,3761	0,2233	0,3718	0,3758	0,2306	
Phr20-Norm	0,9442	0,6236	0,6291	0,3888	0,3921	0,3957	0,2424	0,3919	0,3954	0,2489	0,0715
Phr20-Real	0,9137	0,6420	0,6466	0,4119	0,4150	0,4182	0,2645	0,4148	0,4179	0,2704	
Phr21-Norm	0,9448	0,6589	0,6632	0,4341	0,4368	0,4398	0,2861	0,4367	0,4395	0,2916	0,0580
Phr21-Real	0,9265	0,6731	0,6778	0,4530	0,4561	0,4593	0,3050	0,4559	0,4590	0,3112	
Phr22-Norm	0,8735	0,6917	0,6955	0,4784	0,4809	0,4836	0,3310	0,4809	0,4834	0,3363	0,0903
Phr22-Real	0,8672	0,7156	0,7163	0,5121	0,5125	0,5130	0,3666	0,5125	0,5129	0,3675	
Phr23-Norm	0,8000	0,7358	0,7346	0,5413	0,5403	0,5394	0,3982	0,5402	0,5393	0,3962	0,0813
Phr23-Real	0,8518	0,7512	0,7483	0,5645	0,5619	0,5599	0,4239	0,5620	0,5598	0,4190	
Phr24-Norm	0,8434	0,7716	0,7680	0,5953	0,5927	0,5897	0,4593	0,5927	0,5899	0,4531	0,1167
Phr24-Real	0,8116	0,7969	0,7921	0,6346	0,6312	0,6274	0,5061	0,6308	0,6272	0,4968	
Phr25-Norm	0,8977	0,8183	0,8136	0,6697	0,6656	0,6621	0,5479	0,6653	0,6617	0,5385	0,0918
Phr25-Real	0,9003	0,8355	0,8337	0,6978	0,6965	0,6949	0,5829	0,6959	0,6946	0,5795	
Phr26-Norm	0,8619	0,8532	0,8545	0,7275	0,7288	0,7301	0,6208	0,7287	0,7297	0,6238	0,1103
Phr26-Real	0,8622	0,8715	0,8777	0,7593	0,7648	0,7707	0,6621	0,7644	0,7697	0,6762	
Phr27-Norm	0,9004	0,8892	0,8976	0,7901	0,7982	0,8059	0,7028	0,7978	0,8054	0,7232	0,1012
Phr27-Real	0,8613	0,9059	0,9149	0,8208	0,8285	0,8367	0,7436	0,8283	0,8361	0,7654	
Phr28-Norm	0,8723	0,9233	0,9308	0,8527	0,8593	0,8660	0,7875	0,8589	0,8657	0,8059	0,1194
Phr28-Real	0,9472	0,9405	0,9463	0,8840	0,8896	0,8954	0,8313	0,8894	0,8947	0,8465	
Phr29-Norm	0,9720	0,9563	0,9615	0,9142	0,9194	0,9247	0,8740	0,9194	0,9243	0,8886	0,0888
Phr29-Real	0,9414	0,9717	0,9741	0,9439	0,9461	0,9487	0,9172	0,9459	0,9485	0,9238	
Phr30-Norm	0,8120	0,9861	0,9863	0,9725	0,9723	0,9726	0,9586	0,9724	0,9726	0,9595	0,1071
Phr30-Real	0,8748	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	

Source: Elaborated by the author.

Test oracle definitions in O-FIm/CO require a pre-defined threshold. This threshold is the maximum allowed distance between the features extracted from the model object (in

this case, the audio) and the features extracted from the output produced by the SUT. If this distance is higher than the threshold the oracle consider the executions has failed. In this study, for each test oracle, the threshold was set to be half of the higher standard deviation calculated through the feature results of each test oracle. It is important to highlight that this value is only a guess and, in real testing environments, it can be adjusted by testers in accordance to their own preferences. In cases when the tester does not have any knowledge about the behavior of the feature extractors against her/his dataset, a well know strategy is to define a threshold based on the standard deviation of the features. In addition to that, when the tester uses general feature extractors in experiments, the values of the features can vary severely depending on the sound being evaluated.

From the results collected of the tables shown in Appendix A, we defined five tables with the compilation of results for each test oracle against all of the five subject TTSs. As one can see, these tables (CPqD in Table 20, Ivona in Table 21, ODDCAST in Table 22, VocalWare in Table 23, and VozMe in Table 25) present an overall analysis of our oracles' performance regarding each sentence of each TTS subject.

Table 20 – Overall results for CPqD TTS considering both oracles

CPqD					
Oracle 01			Oracle 02		Human Oracle
	Dist	result	Dist	result	
Sent 01	0,034597	pass	0,019726	pass	pass
Sent 02	0,145556	fail	0,0338	pass	pass
Sent 03	0,155786	fail	0,0281	pass	pass
Sent 04	0,115428	pass	0,0353	pass	pass
Sent 05	0,128378	fail	0,0385	pass	fail
Sent 06	0,512012	fail	0,1710	fail	fail
Sent 07	0,445529	fail	0,0625	fail	fail
Sent 08	0,12403	fail	0,0586	pass	fail
Sent 09	0,367867	fail	0,0881	fail	fail
Sent 10	0,178199	fail	0,0687	fail	pass
Sent 11	0,035898	pass	0,0265	pass	pass
Sent 12	0,019276	pass	0,0424	pass	pass
Sent 13	0,189784	fail	0,0982	fail	fail
Sent 14	0,027858	pass	0,0294	pass	pass
Sent 15	0,152793	fail	0,0686	fail	pass
Sent 16	0,220926	fail	0,0418	pass	pass
Sent 17	0,121607	fail	0,0600	pass	pass
Sent 18	0,15433	fail	0,0795	fail	fail
Sent 19	0,017788	pass	0,0498	pass	pass
Sent 20	0,157371	fail	0,0715	fail	fail
Sent 21	0,174208	fail	0,0580	pass	pass
Sent 22	0,047606	pass	0,0903	fail	fail
Sent 23	0,178734	fail	0,0813	fail	fail
Sent 24	0,091578	pass	0,1167	fail	fail
Sent 25	0,161995	fail	0,0918	fail	fail
Sent 26	0,032927	pass	0,1103	fail	fail
Sent 27	0,078054	pass	0,1012	fail	fail
Sent 28	0,295515	fail	0,1194	fail	fail
Sent 29	0,212901	fail	0,0888	fail	fail
Sent 30	0,204539	fail	0,1071	fail	fail

Source: Elaborated by the author.

Table 24 presents the overall results obtained considering all of the subject TTSs. In this analysis it is possible to see the number of false positives and false negatives of our study. These

Table 21 – Overall results for Ivona TTS considering both oracles

Ivona					
threshold	Oracle 01 0,115002234		Oracle 02 0,065887235		Human Oracle
	Dist	result	Dist	result	
Sent 01	0,098962	pass	0,012147	pass	fail
Sent 02	0,127576	fail	0,019051	pass	pass
Sent 03	0,224751	fail	0,028377	pass	fail
Sent 04	0,068071	pass	0,017602	pass	fail
Sent 05	0,219084	fail	0,074595	fail	fail
Sent 06	0,190256	fail	0,023933	pass	fail
Sent 07	0,457528	fail	0,015158	pass	fail
Sent 08	0,080611	pass	0,022525	pass	fail
Sent 09	0,506826	fail	0,035908	pass	fail
Sent 10	0,256705	fail	0,019372	pass	pass
Sent 11	0,184316	fail	0,021724	pass	pass
Sent 12	0,062418	pass	0,018403	pass	pass
Sent 13	0,182576	fail	0,028203	pass	fail
Sent 14	0,320592	fail	0,010204	pass	pass
Sent 15	0,666129	fail	0,04367	pass	pass
Sent 16	0,138439	fail	0,014242	pass	fail
Sent 17	0,180337	fail	0,012171	pass	pass
Sent 18	0,282415	fail	0,030008	pass	pass
Sent 19	0,267094	fail	0,016336	pass	pass
Sent 20	0,125687	fail	0,026702	pass	fail
Sent 21	0,138522	fail	0,014496	pass	pass
Sent 22	0,111162	pass	0,025725	pass	pass
Sent 23	0,148948	fail	0,022384	pass	pass
Sent 24	0,087679	pass	0,024051	pass	fail
Sent 25	0,235029	fail	0,046382	pass	fail
Sent 26	0,254192	fail	0,021208	pass	pass
Sent 27	0,084096	pass	0,017634	pass	fail
Sent 28	0,127443	fail	0,031415	pass	fail
Sent 29	0,063118	pass	0,019434	pass	pass
Sent 30	0,76132	fail	0,18676	fail	fail

Source: Elaborated by the author.

numbers supporting positively answering our RQ (Are feature-based test oracles able to identify normalization issues in real-world TTSs?).

It can be observed that the final results obtained vary from good to regular depending on the subject TTS. For instance, regarding the VocalWare TTS, the strategy seem to be very useful, since only 9 and 8 false positives/negatives were identified out of the total of evaluations. It is possible to highlight, as well, that depending on the subject TTS, one can reach better results using test oracles based on a large number of features (e.g. Oracle 01). For instance, in the case of the subject TTS VozMe, we can notice that oracle 01 performed better than oracle 02. Disregarding that case, oracle 02 performed better than oracle 01, since to identify slightly differences in a given speech sentence, it is better to use statistical-based feature extractor because: (i) differences in the duration of the sentences; and (ii) need for a high sensibility to detect extra words or wrongs terms that are common in normalization problems.

In order to supporting answer our RQs, we have measured the number of false positives and the number of false negatives obtained in each case. The number of false positives and false negatives is measured from the human-oracles analysis performed in each sentence of each subject TTS. From these numbers, we performed a binomial test (statistical test to measure cases of success) in each test oracle for each TTS subject. Regarding the identification of normalization

Table 22 – Overall results for ODDCAST TTS considering both oracles

ODDCAST					
threshold	Oracle 01		Oracle 02		Human Oracle
	0,074956298		0,018391552		
	Dist	resul	Dist	resul	
Sent 01	0,417429	fail	0,027453	fail	fail
Sent 02	0,245664	fail	0,020336	fail	pass
Sent 03	0,20685	fail	0,038575	fail	pass
Sent 04	0,312324	fail	0,032867	fail	fail
Sent 05	0,147757	fail	0,018728	fail	fail
Sent 06	0,530428	fail	0,05255	fail	fail
Sent 07	0,368435	fail	0,029693	fail	fail
Sent 08	0,071106	pass	0,01907	fail	fail
Sent 09	0,194873	fail	0,00974	pass	fail
Sent 10	0,292361	fail	0,043325	fail	pass
Sent 11	0,330828	fail	0,021465	fail	pass
Sent 12	0,174469	fail	0,018644	fail	pass
Sent 13	0,138408	fail	0,032772	fail	fail
Sent 14	0,112579	fail	0,009847	pass	pass
Sent 15	0,043727	pass	0,016388	pass	pass
Sent 16	0,263821	fail	0,026625	fail	pass
Sent 17	0,254375	fail	0,015577	pass	pass
Sent 18	0,117799	fail	0,019714	fail	pass
Sent 19	0,104055	fail	0,013277	pass	pass
Sent 20	0,299725	fail	0,021055	fail	pass
Sent 21	0,10468	fail	0,01046	pass	pass
Sent 22	0,180614	fail	0,017291	pass	fail
Sent 23	0,271264	fail	0,009912	pass	fail
Sent 24	0,085947	fail	0,015543	pass	pass
Sent 25	0,050871	pass	0,014164	pass	pass
Sent 26	0,104097	fail	0,015066	pass	pass
Sent 27	0,095525	fail	0,012778	pass	fail
Sent 28	0,232121	fail	0,026317	fail	fail
Sent 29	0,233105	fail	0,01125	pass	fail
Sent 30	0,221131	fail	0,0269	fail	fail

Source: Elaborated by the author.

problems in TTS systems, this deeper analysis of the results allowed us to notice several facts about our technique:

1. There is no “gold-standard” oracle configuration for all subject TTSs: The results of this study revealed that some oracle configurations can work properly to a subject TTS and poorly to another. Since we used the same 30 sentences in our study, we can take Oracle O1 to illustrate the variability of efficiency of the test oracles. First, Oracle O1 has performed well to identify normalization problems in VozMe, with a statistical significance of 0.0002. However, the same oracle configuration against the same 30 sentences has poorly evaluated the sentences generated by ODDCAST, performing pretty similar to a random pass/fail classifier. A deep study can be conducted in such cases in order to identify the reason of such differences. Maybe, another threshold strategy or even changing the whole oracle configuration could help improving the oracle performance;
2. Threshold adjusts can improve the results significantly: defining threshold values in feature-based test oracles is always a domain-specific activity. Besides the flexibility of the test oracles implemented through O-FIm/CO, several aspects can be fundamental to change the regular values of the features. In sounds, these factors are many: TTS language,

Table 23 – Overall results for VOCALWARE TTS considering both oracles

VOCALWARE					
threshold	Oracle 1		Oracle 02		Human
	0,146655468		0,020868628		
	Dist	result	Dist	result	Oracle
Sent 01	0,204957	fail	0,009602	pass	fail
Sent 02	0,123552	pass	0,010707	pass	pass
Sent 03	0,308909	fail	0,039278	fail	fail
Sent 04	0,121626	pass	0,020427	pass	pass
Sent 05	0,239905	fail	0,006342	pass	fail
Sent 06	0,172746	fail	0,030211	fail	fail
Sent 07	0,359603	fail	0,011912	pass	fail
Sent 08	0,136772	pass	0,0261	fail	fail
Sent 09	0,610367	fail	0,032802	fail	fail
Sent 10	0,127168	pass	0,012733	pass	pass
Sent 11	0,306249	fail	0,008089	pass	pass
Sent 12	0,118267	pass	0,0403	fail	pass
Sent 13	0,247072	fail	0,043082	fail	fail
Sent 14	0,288382	fail	0,006622	pass	pass
Sent 15	0,201393	fail	0,017753	pass	pass
Sent 16	0,394069	fail	0,025286	fail	fail
Sent 17	0,418786	fail	0,014771	pass	pass
Sent 18	0,383346	fail	0,022523	fail	fail
Sent 19	0,305596	fail	0,012272	pass	pass
Sent 20	0,681484	fail	0,059697	fail	fail
Sent 21	0,08764	pass	0,043277	fail	fail
Sent 22	0,233099	fail	0,012565	pass	fail
Sent 23	0,604191	fail	0,014395	pass	pass
Sent 24	0,078805	pass	0,013829	pass	pass
Sent 25	0,453188	fail	0,017195	pass	fail
Sent 26	0,507323	fail	0,013198	pass	pass
Sent 27	0,292719	fail	0,017932	pass	fail
Sent 28	0,535078	fail	0,019539	pass	fail
Sent 29	0,221172	fail	0,02229	fail	fail
Sent 30	0,215397	fail	0,023014	fail	fail

Source: Elaborated by the author.

TTS implementation (concatenative or parametric), voice genders, sound frequencies, noises, etc. In this case, deeper studies are necessary in order to mitigate the variation of features extracted from different TTSs. All of these initiatives should be conducted to drive suggestions of better threshold adjustment. However, in real testing scenarios, a tester can define strategies to identify better threshold adjustments for her/his own TTS, since the tester will be dealing always with the same TTS. On the other hand, when a practitioner has several TTS implementations, threshold issues will be always a challenging due to the need for maintenance in testing scripts;

3. The oracle problem (false positives and false negatives) is a remaining problem: in our study, through the human oracle analysis, we observed some specific cases in which both test oracles strategies fail on identifying normalization problems, configuring a typical case of false negative. These cases were more common in Oracle 2 and to identify the cause is a non-trivial task. The application of some segmentation technique or even an signal alignment algorithm can mitigate those cases; and
4. Feature-based test oracles is a promising strategy of test automation to identify normalization problems in TTS applications: The overall analysis of this study has shown that feature-based test oracles can be applied effectively to help alleviating a sub-problem of

Table 24 – General results and metrics collected from the study

TTS	Metric	Oracle 01 (O1)	Oracle 02 (O2)	P-Value O1	P-Value O2
CPqD	False Positive	7	2	0.1002	<than 0.0001
	False Negative	4	2		
	True Positive	13	15		
	True Negative	6	11		
Ivona	False Positive	11	0	0.4278	0.4278
	False Negative	5	14		
	True Positive	11	2		
	True Negative	3	14		
ODDCAST	False Positive	14	8	0.5722	0.2923
	False Negative	1	5		
	True Positive	13	9		
	True Negative	2	8		
VocalWare	False Positive	7	1	0.0214	0.0081
	False Negative	2	7		
	True Positive	16	11		
	True Negative	5	11		
VozMe	False Positive	4	5	0.0002	0.4278
	False Negative	1	9		
	True Positive	15	7		
	True Negative	10	9		

Source: Elaborated by the author.

practitioners – the normalization issues. Once a testing environment is defined, through O-FIm/CO, practitioners can implement test scripts to regularly check normalization issues in several cases: (i) after system’s maintenance, (ii) before the delivery of a specific TTS in a given reduced database, and (iii) after implementing a novel normalization rule. We highlight that regularly, normalization problems in TTS applications are tested mainly through two strategies: (i) Using test cases implemented through “diff” algorithms applied the pre-processing algorithm of the TTS. Which means, there is no systematic strategy applied, only a checker that simulates the behavior of the text normalizer in some situations, and (ii) though informal, subjective, and ad-hoc listening tests. In fact, depending on the application, a group of normalization rules are included in a TTS, bringing the necessity for new verification/validation activities. Then, to conclude, we believe that effective feature-based test oracles can substitute or, even, complement these two testing strategies.

5.3.5 Threats to Validity

The first threat to the validity of the presented results is the use of a demo online application for the experiment that could affect the “external validity” of the results. However, since the feature-based test oracles implemented through O-FIm/CO abstracts the system’s logic into a black-box, the results are perceived general for all TTS subject. Still regarding the external validity, a greater threat is that we performed our evaluation only based on female Brazilian Portuguese voices. This is due to the fact Brazilian Portuguese is our native language and, then, it is more suitable to perform human oracle analysis to support collecting the false positive/negative numbers. The choice of female voices is due to the fact that our industry

Table 25 – Overall results for VozMe TTS considering both oracles

VozMe					
threshold	Oracle 01		Oracle 02		Human Oracle
	0,207252584		0,025718112		
	Dist	result	Dist	result	
Sent 01	0,394254	fail	0,072822	fail	fail
Sent 02	0,402066	fail	0,026279	fail	fail
Sent 03	0,395806	fail	0,007997	pass	fail
Sent 04	0,3871	fail	0,047618	fail	fail
Sent 05	0,1789	pass	0,03306	fail	pass
Sent 06	0,258858	fail	0,004724	pass	fail
Sent 07	0,16855	pass	0,016334	pass	pass
Sent 08	0,212517	fail	0,013996	pass	fail
Sent 09	0,290191	fail	0,040953	fail	fail
Sent 10	0,086234	pass	0,006455	pass	pass
Sent 11	0,356473	fail	0,058439	fail	fail
Sent 12	0,247816	fail	0,019645	pass	fail
Sent 13	0,541464	fail	0,039008	fail	fail
Sent 14	0,110842	pass	0,019596	pass	pass
Sent 15	0,41468	fail	0,016651	pass	pass
Sent 16	0,265793	fail	0,00812	pass	fail
Sent 17	0,532578	fail	0,01421	pass	fail
Sent 18	0,20045	pass	0,007997	pass	pass
Sent 19	0,349185	fail	0,013787	pass	fail
Sent 20	0,123077	pass	0,009893	pass	pass
Sent 21	0,434233	fail	0,047329	fail	pass
Sent 22	0,206792	pass	0,073217	fail	fail
Sent 23	0,231415	fail	0,016073	pass	pass
Sent 24	0,501566	fail	0,012648	pass	pass
Sent 25	0,123644	pass	0,042878	fail	pass
Sent 26	0,129516	pass	0,04471	fail	pass
Sent 27	0,137718	pass	0,019439	pass	pass
Sent 28	0,171466	pass	0,0546	fail	pass
Sent 29	1,301724	fail	0,017369	pass	fail
Sent 30	0,366101	fail	0,009585	pass	fail

Source: Elaborated by the author.

partners (CPqD) has real-world TTSs implemented only through female readers. Then, they were particularly interested in results of test oracles evaluating females voices.

Another threat is the choice of online TTS applications. Since we are dealing with some web service, we can not assure that connection and communication problems can affect (or not) the final audio output. In the case of CPqD TTS, we collected our data in a desktop pre-installed version of the system. For all other TTSs (Ivona, ODDCAST, VocalWare, VozMe) we collect their data through the software audacity. Then, to mitigate problems, we segmented the silences at the end and at the beginning of the audio signals. This process was made to improve the feature extraction process, avoiding dealing with unwanted situations such as, extracting features from silences, etc. However, since the threshold strategy was made equally to all TTS (based on the standard deviation of the features extracted), the results of our study mitigates this threat.

The results of this work indicate that a combination of the two or three threshold definition strategies would improve the test oracle efficacy. This factor (changing and combining threshold strategies) is not explicitly evaluated in this study and we can only mention that the overall results indicates favorable scenarios and better results when changing the threshold strategy based on the TTS application. In this study, there is no evidence for it, however in future approaches we can state different threshold strategies to evaluate their performance against the same data. Then,

this study can be considered a reference and a starting point to future evaluations.

Finally, following the four-step standard threat analysis, the threats identified in this study are:

Internal validity: This empirical study was exploratory in nature but also focused with a narrow scope, driven by a well defined research question. Then, the internal validity of this study is considered high;

External validity: As we mentioned before, the collection through the software audacity and the wide usage of online TTS applications are the major limiting factors of the generalization of our study;

Construct validity: as the feature-based test oracles are able to test TTS application using only their output, the construct validity in this case is considered high. In addition to that, the complexity (regarding normalization issues) of the 30 sentences evaluated is considered high, contributing to mitigate this threat;

Conclusion validity: We have presented the methodology in full and showed all study results, i.e. tables and some algorithms. These results support our conclusions and we therefore claim that we have high conclusion validity.

5.3.6 Final Remarks and Findings

Through this study we have demonstrated how feature-based test oracles can perform against a real-world complex-output testing problem – normalization defects in TTS applications. We believe that further domain-specific analysis can increase the effectiveness of the test oracles on identifying normalization inconsistencies. It is important to highlight that industry cooperators from CPqD are studying the used of this approach in practice to help evaluating normalization problems in their solution.

The main findings collect from this study are three: (i) feature-based test oracles can effectively support automated testing activities of real-world TTS systems; (ii) the performance of different test oracle configuration on detecting actual normalization problems varies from one TTS to another; and (iii) domain specific implementations, threshold adjustments, and alignments algorithms can be associated to feature-based test oracles to improve their performance.

5.4 Feature-based test oracles to evaluate synthesized speech from TTSs with reduced audio databases

Another sub-problem of TTS systems is the dependency of a huge database of recorded speech to produce high-quality synthesized voices. During the process of creating synthesized speech, after text normalization and tokenization processes, a TTS solution goes through a

process that consists of searching, in an audio database, for recorded speeches to address each token identified. In general, wide and complete audio databases support the creation of suitable and high-quality voices. On the other hand, reduced and small audio databases can lead poorly and low-quality speech. This is because when a given TTS does not find any recorded audio reference in its database to a particular token, some algorithm is required to synthesize a corresponding speech, creating artificial voices.

The effective use of TTS applications in mobile and ubiquitous systems is severely limited by the database size. When the first TTS applications were created, all of them were desktop-based. Then, the size of audio databases was considered a minor problem for TTS applications. However, contemporary application are ubiquitous and, generally, they are also designed to work properly in mobile platforms and embedded systems. Parametric TTSs, that are more portable and light than the concatenative ones, alleviate these issues. However the databases' size is a limiting factor to the wide usage of TTS in mobile/embedded applications. Further, this is one of the reasons that makes the TTS applications being common as an online service on the web. Generally, a huge database is stored in a server, so an online TTS can request their services to synthesizes speech.

When implementing some embedded/mobile TTS, practitioners perform subjective and ad-hoc listening tests to define an acceptable relation between the size of the audio database of the TTS under implementation and its quality. For example, depending on the TTS's purpose, starting from an acceptable solution, practitioners can implement several versions of that solution with reduced audio databases. Then, through ad-hoc listening tests, some human subjects evaluate whether the reduced versions produce acceptable speech regarding their purposes.

Given this introductory scenario, this study addresses to raise evidences about the usage of feature-based test oracles and O-FIm/CO by practitioners to implement a testing environment to synthesized speech produced by TTS systems with reduced audio databases. Regularly, this experiment is addressed to check whether feature-based test oracles are able to effectively identify speeches' quality deviations caused by reduced audio databases. In this sense, this experiments' goals are a threefold: (i) raise evidences on how feasible is the usage of feature-based test oracles to automatically check the quality of speeches synthesized by TTSs with audio reduced databases; (ii) evaluate the trade-off on using feature-based test oracles to alleviate solving a domain-specific problem of system with complex output; and (iii) evaluate the behavior of different oracle configurations with threshold strategies.

It is important to highlight that this empirical validation was designed in consonance of a real-world problem faced by practitioners when implementing TTS applications in mobile applications and embedded systems. Once again, CPqD collaborated to design this experiment in a real-world industry environment. Finally, our objects of study are the designed test oracles and their different configuration with different threshold. The perspective of this study is from the testers' point of view and, then, the effect studied is associated with the effective-

ness and reliability of the test oracles, considering number of false positive/negatives and true positives/negatives.

5.4.1 Research Questions

This experiment is motivated by a single Research Question:

RQ1 Are feature-based test oracles able to perform effective tests in speech produced using TTSs with reduced audio databases?

The metrics used to support answering this RQ1 are the number of false positives/negatives and true negatives/positives produced by the test oracles.

5.4.2 TTS Subject

CPqD TTS is the subject used in this study. To address this experiments' goals, four different audio databases were associated to CPqD TTS. The next subsection presents further information about the procedure and design defined towards answering RQ1.

5.4.3 Experimental Design and Procedure

The experimental strategy defined to raise evidences to answer our RQ1 is:

- Creating four different versions of CPqD TTS with four different audio databases' size. Then, in collaboration with industry practitioners from CPqD, we have obtained four audio databases to CPqD TTS: (i) 968 MegaBites (MB), (ii) 251 MB, (iii) 75 MB, and (iv) 30 MB.

To make this study easier to understand, starting from here, we have named each version of the CPqD TTS as follows: "Max" (968Mb), "Md" (251Mb), "Min" (75Mb), and "MMin"(30Mb);

- Recording 30 different sentences using each one of the four versions of the CPqD TTSs mentioned above (Max, Md, Min, and MMin), creating a dataset of 120 audio signals. To do so, we explored the same 30 sentences previously presented in Table 17;
- Adjusting five feature-based oracle configuration with three different threshold definition to automatically evaluate the sentences regarding their quality. Details about the five oracle configuration are presented below;
 - *Oracle 01*: This test oracle is addressed to check the effectiveness of feature-based test oracle with different single dimensional feature extractors. Then, this test oracle is set with 17 feature extractors: (1) Spectral Centroid, (2) Spectral Rolloff Point, (3)

Spectral Flux, (4) Compactness, (5) Spectral Variability, (6) Root Mean Square, (7) Fraction Of Low Energy Windows, (8) Zero Crossings, (9) Strongest Beat, (10) Beat Sum, (11) Strength Of Strongest Beat, (12) Strongest Frequency Via Zero Crossings, (13) Strongest Frequency Via Spectral Centroid, (14) Strongest Frequency Via FFT, (15) Partial Based Spectral Centroid, (16) Peak Based Spectral Smoothness, and (17) Relative Difference Function. Implementation details and the specification of each feature extractor can be found online¹²;

- *Oracle 02*: The second oracle configuration we uses a single multidimensional feature extractor called Traditional Area Method of Moments, previously explored in the experiment presented before. This feature extractor uses 2D statistical methods grouped in ten dimensions to check different aspects of the audio signal. The choice for this feature extractor is due to the success of its application in previous studies;
 - *Oracle 03*: This third oracle configuration is composed by the multidimensional feature extractor called Chroma. This feature extractor has 12 dimensions that are designed to capture the harmonic signature of spectra from a signal. We justify the choice of this feature extractor because we believe the reduced audio databases directly affects the signature of the audio signal;
 - *Oracle 04*: This forth oracle is designed using the feature extractor Linear Prediction Coefficients (LPC) calculated using autocorrelation and Levinson-Durbin recursion, that is a recursive linear algebra algorithm to extract features from specified. To do so, nine dimensions are used by the feature extractor. The choice by this feature extractor to compose oracle 04 has the same justification presented to oracle 03;
 - *Oracle 05*: The last oracle configuration addresses to explore the multidimensional feature extractor called MFCC. This feature extractor uses a power spectrum algorithm that approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. The choice of this feature extractor is because it tries to simulate the human auditory systems; and
 - Euclidean distance was used in all of the test oracles. Mean was the aggregator factor used in single-multidimensional feature extractors.
- Finally, the threshold definition for this experiment, such as in experiments presented before, we used a standard deviation-based strategy. Then, three threshold were defined: (i) **t1** the double of the maximum standard deviation among results of the feature extractors' (dimensions when multidimensional feature extractors are considered); (ii) **t2** the maximum standard deviation among the results of the feature extractors; and (iii) **t3** half of the maximum standard deviation among the results of the feature extractors.

¹² see: [www.labes.icmc.usp.br/\\$\sim\\$Rpaes/dissertation.html](http://www.labes.icmc.usp.br/$\sim$Rpaes/dissertation.html)

After following the aforementioned steps, we loaded the feature-based oracle configurations against the recorded dataset. For each one of the 30 sentences, the audio signal collected with the largest database was considered the model to the test oracles. Then, we have measured the distance (difference) of the speech generated from the larger database against the other three smaller versions. We considered a window size of 64 and 0.2 as the overlap between the windows. Next, we present the results of this experiment.

We carried out a manual analysis over our dataset to identify the general effects of the reduced audio databases in the quality of the speech produced. Our findings through this analysis are:

- “*Max*” (968Mb): the speeches produced by this CPqD TTS version are high-quality and easy to understand. The quality of the speeches should be fine for several applications due to the naturalness of them. Intonation and prosody aspects make the speech sound like a real human reading the sentences;
- “*Md*” (251Mb): in general, the sentences produced with this CPqD TTS version are very similar to the “*Max*” version. Besides the fact that the audio database is reduced almost three times, the speech produced is fair enough to be used in several TTS application. Through our listening tests, for all of the 30 sentences, We did not notice significant losses in comparison the “*Max*”;
- “*Min*” (75Mb): this version of CPqD TTS is attached to an audio database that is almost 13 times smaller than “*Max*”. Then, though listening tests, we noticed some differences in the soundness of the speech produced in this study. Mainly, two different effects were noticed: (i) “clicks”; (ii) faster sentences. However, we considered the speech acceptable for TTS applications for two reasons: (i) all of the 30 sentences were clearly understood by us in the very first time we listened them; and (ii) the soundness of the speech produced is good enough to use this CPqD TTS version in several applications, once the naturalness of the voice is preserved;
- “*MMin*”(30Mb): this last version of CPqD TTS has an audio database that is 32 times smaller than “*Max*”. Besides to be small enough to several mobile/embedded applications, the quality of the speech produced is severely limited. Through listening tests we noticed three quality problems: (i) concatenation problems that make different words to be read together, with no pause; (ii) missing syllables that lead the speech final to be without some words; and (iii) prosody (intonation, tone, stress, and rhythm) problems that difficult the easy understanding of all sentences. In some cases, changes in the stressed syllable of the word, unnecessary intonation, and unexpected tones make some sentences very low-quality and poor. Then, we considered all of the sentences produced by this version of CPqD unacceptable to be explored in real-world TTS applications.

Once we have manually evaluated all of the speech produced, we carried out this empirical evaluation making the following assumptions: (i) the speeches produced by “Max” are considered the standard reference to our tests, so the test oracles were designed to use this configuration as their reference; and (ii) we assume that the test oracles should accept (pass) all of the speeches produced by “Md”, and “Min” versions. Finally, all of the speeches produced by “MMin” should be rejected (fail). Table 26 presents all of the expected verdicts from the test oracles implemented in this study. Then, from Table 26 we figured the number of false results (false negatives/positives) of this study. Next, Section 5.4.4 presents results collected from the study described here.

Table 26 – Template with expected verdicts for automated test oracles

TTS version	“Max”	“Md”	“Min”	“MMin”
Expected Oracle Verdict	pass	pass	pass	fail

5.4.4 Results and Discussion

Table 27 all of the results, including all five test oracle configurations with three different threshold strategy. All of the pass/fail verdicts of the feature-based test oracles are presented. The results are presented by sentence (row) associated with each oracle configuration (column). The first column presents the sentences collected from some version of the CPqD TTS. For example, 01Max, 01Md, 01Min, and 01MMin represent the Sentence 01 (Table 17) collected using “Max”, “Md”, “Min”, and “MMin” respectively.

Table 27 – Overall results of all five test oracles configuration with three different thresholds

	Oracle 01			Oracle 02			Oracle 03			Oracle 04			Oracle 05		
	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3
	.356	.178	.089	.134	.067	.033	.169	.084	.042	.655	.327	.164	.650	.325	.163
01Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
01Md	fail	fail	fail	pass	pass	pass	pass	fail	fail	pass	pass	fail	pass	fail	fail
01Min	pass	pass	fail	pass	pass	pass	pass	pass	fail	pass	fail	fail	pass	fail	fail
01MMin	fail	fail	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail
02Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
02Md	pass	fail	fail	pass	pass	pass	fail	fail	fail	pass	fail	fail	pass	fail	fail
02Min	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
02MMin	fail	fail	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail
03Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
03Md	pass	pass	pass	pass	pass	pass	pass	fail	pass	pass	pass	fail	pass	pass	fail
03Min	pass	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	pass	fail	fail	fail
03MMin	fail	fail	fail	pass	pass	pass	pass	fail	fail	pass	fail	fail	pass	fail	fail
04Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
04Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
04Min	fail	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	fail	pass	fail	fail
04MMin	fail	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
05Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
05Md	fail	fail	fail	pass	fail	fail	pass	fail	pass	fail	fail	fail	fail	fail	fail
05Min	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail	fail	fail	fail
05MMin	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail	fail	fail	fail

Continued on next page

Table 27 – continued from previous page

	Oracle 01			Oracle 02			Oracle 03			Oracle 04			Oracle 05		
	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3
	.356	.178	.089	.134	.067	.033	.169	.084	.042	.655	.327	.164	.650	.325	.163
06Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
06Md	pass	pass	fail	pass	pass	pass	pass	fail	pass	pass	pass	fail	pass	pass	fail
06Min	fail	fail	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
06MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
07Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
07Md	fail	fail	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
07Min	fail	fail	fail	fail	fail	fail	fail	pass	fail	pass	fail	fail	pass	fail	fail
07MMin	fail	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	fail	fail	fail
08Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
08Md	pass	fail	fail	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail
08Min	fail	fail	fail	pass	pass	fail	pass	pass	pass	pass	fail	fail	fail	fail	fail
08MMin	fail	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	fail	fail	fail
09Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
09Md	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	pass	pass	pass	pass	pass
09Min	pass	fail	fail	pass	pass	fail	fail	pass	fail	pass	fail	fail	fail	fail	fail
09MMin	fail	fail	fail	pass	pass	pass	fail	fail	fail	fail	fail	fail	fail	fail	fail
10Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
10Md	pass	pass	fail	pass	pass	pass	pass	fail	fail	pass	fail	fail	pass	pass	fail
10Min	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
10MMin	pass	fail	fail	pass	fail	fail	pass	fail	fail	pass	pass	fail	fail	fail	fail
11Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
11Md	pass	fail	fail	pass	pass	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
11Min	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail
11MMin	fail	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
12Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
12Md	pass	pass	pass	pass	pass	pass	pass	fail	fail	pass	pass	pass	pass	pass	fail
12Min	pass	fail	fail	pass	pass	fail	fail	pass	fail	pass	pass	fail	pass	fail	fail
12MMin	pass	pass	fail	pass	pass	pass	fail	fail	fail	pass	pass	pass	pass	fail	fail
13Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
13Md	fail	fail	fail	fail	fail	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
13Min	fail	fail	fail	pass	fail	fail	fail	pass	fail	pass	fail	fail	pass	fail	fail
13MMin	fail	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
14Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
14Md	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
14Min	pass	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	pass	pass	fail	fail
14MMin	pass	fail	fail	pass	pass	fail	pass	fail	pass	pass	pass	fail	pass	fail	fail
15Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
15Md	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
15Min	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
15MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	fail	fail	fail
16Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
16Md	pass	fail	fail	pass	pass	pass	pass	fail	fail	pass	pass	fail	pass	fail	fail
16Min	fail	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	fail	pass	pass	fail
16MMin	fail	fail	fail	pass	pass	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
17Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
17Md	pass	pass	fail	pass	pass	pass	pass	fail	pass	pass	pass	pass	pass	fail	fail
17Min	pass	fail	fail	pass	pass	fail	pass	pass	fail	pass	pass	fail	fail	fail	fail
17MMin	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail	fail	fail	fail
18Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
18Md	pass	pass	fail	pass	pass	fail	pass	fail	pass	pass	pass	fail	pass	fail	fail
18Min	fail	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	fail	pass	fail	fail
18MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	fail	fail	fail
19Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
19Md	pass	pass	pass	pass	pass	pass	pass	fail	fail	pass	pass	fail	pass	pass	fail

Continued on next page

Table 27 – continued from previous page

	Oracle 01			Oracle 02			Oracle 03			Oracle 04			Oracle 05		
	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3	t1	t2	t3
	.356	.178	.089	.134	.067	.033	.169	.084	.042	.655	.327	.164	.650	.325	.163
19Min	pass	fail	fail	pass	fail	fail	fail	pass	fail	pass	pass	fail	pass	fail	fail
19MMin	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
20Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
20Md	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail
20Min	fail	fail	fail	pass	fail	fail	pass	pass	pass	pass	fail	fail	pass	fail	fail
20MMin	pass	fail	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail	fail	fail	fail
21Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
21Md	pass	fail	fail	pass	pass	pass	pass	fail	fail	pass	fail	fail	pass	pass	fail
21Min	pass	pass	fail	pass	fail	fail	fail	pass	fail	pass	pass	fail	pass	fail	fail
21MMin	pass	pass	fail	pass	pass	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
22Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
22Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	pass	pass	pass	fail
22Min	pass	fail	fail	fail	fail	fail	fail	pass	fail	pass	fail	fail	pass	fail	fail
22MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
23Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
23Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail
23Min	fail	fail	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail	fail	fail	fail
23MMin	fail	fail	fail	pass	pass	fail	pass	fail	pass	fail	fail	fail	fail	fail	fail
24Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
24Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail
24Min	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
24MMin	pass	fail	fail	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
25Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
25Md	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	fail	fail
25Min	fail	fail	fail	fail	fail	fail	pass	pass	pass	pass	pass	fail	pass	fail	fail
25MMin	fail	fail	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail	pass	fail	fail
26Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
26Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
26Min	pass	fail	fail	fail	fail	fail	pass	pass	pass	pass	pass	fail	pass	fail	fail
26MMin	pass	pass	fail	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail
27Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
27Md	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	pass	pass
27Min	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	pass	fail	pass	fail	fail
27MMin	fail	fail	fail	pass	fail	fail	pass	fail	pass	pass	pass	fail	pass	fail	fail
28Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
28Md	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
28Min	pass	fail	fail	fail	fail	fail	fail	pass	fail	pass	fail	fail	pass	fail	fail
28MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	fail	fail	pass	fail	fail
29Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
29Md	pass	pass	fail	pass	pass	fail	pass	fail	fail	pass	pass	fail	pass	pass	fail
29Min	pass	fail	fail	pass	fail	fail	pass	pass	fail	pass	fail	fail	pass	fail	fail
29MMin	pass	fail	fail	pass	fail	fail	fail	fail	fail	pass	pass	fail	fail	fail	fail
30Max	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass	pass
30Md	pass	pass	fail	pass	pass	fail	pass	fail	pass	pass	pass	fail	pass	pass	fail
30Min	pass	fail	fail	fail	fail	fail	pass	pass	fail	pass	pass	pass	fail	fail	fail
30MMin	pass	fail	fail	pass	fail	fail	pass	fail	pass	pass	pass	fail	fail	fail	fail

Source: Elaborated by the author.

Considering the template presented in Table 26 in comparison with the actual results, it is possible to notice that the results achieved are diverse. Through a global analysis of the results,

one can identify mainly three different cases:

- *Case 01* – when the oracle configuration and the threshold strategy lead to high number of false results. In these cases, false positives and false negatives are predominant and, then, the configuration is useless for testing activities. Nominally, five combinations of oracles and threshold could be included in this case: (i) Oracle 01 + t3; (ii) Oracle 02 + t3; (iii) Oracle 03 + t3; (iv) Oracle 04 + t3; and (v) Oracle 01 + t3. Out of 90 verdicts, these combinations generated false results ranging from 51 to 59 times. With these numbers, we can define that these combinations are not applicable to the problem of identifying quality issues in audio produced by reduced audio database. For instance, taking the Oracle 03 with t3 to evaluate Sentence 14, it is possible to observe that the oracle produced three wrong verdicts out of three analysis;
- *Case 02* – when the oracle configuration combined with some threshold strategy performs tests similar to a random classifier. In these cases, the combination between oracle configuration and threshold produce results that cannot be trustful enough to be used in productive systematic testing environments. Oracles in this case produced a similar number of correct and incorrect verdicts. We observed four combinations of oracles in this situation: (i) Oracle 01 + t2; (ii) Oracle 02 + t1; (iii) Oracle 02 + t2; and (iv) Oracle 05 + t2. To illustrate this case, one can consider Oracle 05 with the threshold strategy t2 during the evaluation of the sentences 3 and 4. Out of six analysis, the oracle produced three false verdicts. Oracles in this cases are considered low-effectiveness, being useless in effective testing environments; and
- *Case 03* – when the combination between oracle and threshold produces high number of correct verdicts. In these cases, the combination of test oracle and threshold are reliable enough to be effectively used in real-world testing environments, having high statistical significance. Analyzing our study's results, Six combinations of test oracle and threshold led this case: (i) Oracle 01 + t1; (ii) Oracle 03 + t1; (iii) Oracle 03 + t2; (iv) Oracle 04 + t1; (v) Oracle 04 + t2; and (vi) Oracle 05 + t1. For example, one can highlight Oracle 01 with t1 on testing sentences 2 and 3. In both cases, the combination achieved 100% of right verdicts. We highlight that oracles considered in this cases are fair enough to be effectively explored in testing environments.

Given this scenario, we can highlight that feature-based test oracle can be effectively explored to implement systematic and automated testing environments to evaluate the quality of speeches produced in TTSs with reduced audio databases.

To better analysis this experiment's results, a single group of hypothesis was extracted from RQ1 (Sec. 5.4.1). So, this hypothesis addresses to the effectiveness of the feature-based test oracles on composing testing environments to TTSs with reduced audio database. Then,

the group of hypotheses defined evaluate the number false verdicts (nominally **FP/N**) and true verdicts (nominally **TP/N**) of the oracle combinations. The null and alternative hypotheses are listed below.

Null hypothesis (H_0):

“There is no difference between the number of correct (**TP/N**) and incorrect (**FP/N**) verdicts produced by feature-based test oracles”.

Alternative hypothesis (H_1):

“The percentage of correct verdicts (**TP/N**) produced by feature-based test oracles is greater than incorrect (**FP/N**) verdicts”.

To get statistical evidence of the effectiveness on using feature-based test oracles to test TTS systems with reduced audio databases, we used the one-tailed binomial test with a statistical significance level $\alpha = 0.05$. Once again, the reason that we used binomial test was because it is recommended to check statistical evidences from data if binary outcomes (cases of success). In the case of test oracles, considering the number of false verdicts, binomial tests can show the statistical significance of deviations from an expected distribution of observations into two categories (fail and pass). The numbers are based on 90 trials, since we had 30 sentences and three TTS versions with reduced audio database.

Our statistical analysis has revealed that six out fifteen test oracle combination had positive results with statistical significance. *Oracle 01 with t1*, *Oracle 03 with t1*, *Oracle 03 with t2*, *Oracle 04 with t1*, *Oracle 04 with t2*, and *Oracle 05 with t1* are the combinations that presented statistical significance. In all of these cases, the statistical evidences contributed to reject H_0 and showed the effectiveness on using feature-based test oracles.

Table 28 presents all of the statistical analysis carried out from the numbers collected in this study. The table presents a global analysis including the number of false verdicts (**FP/N**), number of true verdicts (**TP/N**) and the statistical test for H_0 based on these numbers. Thus, a brief analysis with the meaning of each statistical result is carried out.

In the meaning column of Table 28, it is presented a deep interpretation of what the numbers revealed about the results. In these interpretations, it is presented the probability of the main events happen in a particular oracle combination. These interpretations are based in the concepts of the one-tail binomial test, so they describe the chance of the well-succeed events happens in three different occasions: (i) exactly **TP/N** times out of 90 tests; (ii) fewer than **TP/N** out of 90 tests; and (iii) more than **TP/N** out of 90 tests.

To give a graphical and informative information about the performance of each oracle configuration, in consonance with the statistical tests, Figure 30¹³ presents the binomial distribution for each test oracle configuration. The binomial distribution represents the statistical

¹³ This figure was generated from the applications created by Matt Bognar and Iowa University. See: <http://homepage.divms.uiowa.edu/~mbognar/applets/bin.html>

Table 28 – Statistical analysis of feature-based test oracles to evaluate TTSs' outputs with small audio databases

Or.	T.	FP/N	TP/N	p-value	H ₀	meaning
Oracle 01	t1	32	58	0,004204	Rej.	The prob. of exactly 58 out of 90 is p = 0.00194041
						The prob. of exactly, or fewer than, 58 out of 90 is p = 0.99791744
						The prob. of exactly, or more than, 58 out of 90 is p = 0.00402296
						The (one-tailed) prob. of exactly, or greater than, 58 out of 90 is p = 0.004204
	t2	47	43	0,700794	Acc.	The prob. of exactly 43 out of 90 is p = 0.07681074
						The prob. of exactly, or fewer than, 43 out of 90 is p = 0.3760166
						The prob. of exactly, or more than, 43 out of 90 is p = 0.70079414
						The (one-tailed) prob. of exactly, or fewer than, 43 out of 90 is p = 0.375915
	t3	57	33	0,995977	Acc.	The prob. of exactly 33 out of 90 is p = 0.00341041
						The prob. of exactly, or fewer than, 33 out of 90 is p = 0.00743338
						The prob. of exactly, or more than, 33 out of 90 is p = 0.99597704
						The (one-tailed) prob. of exactly, or fewer than, 33 out of 90 is p = 0.007667
Oracle 02	t1	37	53	0,056923	Acc.	The prob. of exactly 53 out of 90 is p = 0.02039741
						The prob. of exactly, or fewer than, 53 out of 90 is p = 0.96372523
						The prob. of exactly, or more than, 53 out of 90 is p = 0.05667218
						The (one-tailed) prob. of exactly, or greater than, 53 out of 90 is p = 0.056923
	t2	39	51	0,123126	Acc.	The prob. of exactly 51 out of 90 is p = 0.03793203
						The prob. of exactly, or fewer than, 51 out of 90 is p = 0.9148788
						The prob. of exactly, or more than, 51 out of 90 is p = 0.12305322
						The (one-tailed) prob. of exactly, or greater than, 51 out of 90 is p = 0.123126
	t3	52	38	0,943328	Acc.	The prob. of exactly 38 out of 90 is p = 0.02844902
						The prob. of exactly, or fewer than, 38 out of 90 is p = 0.0851212
						The prob. of exactly, or more than, 38 out of 90 is p = 0.94332782
						The (one-tailed) prob. of exactly, or fewer than, 38 out of 90 is p = 0.085293
Oracle 03	t1	33	57	0,007667	Rej.	The prob. of exactly 57 out of 90 is p = 0.00341041
						The prob. of exactly, or fewer than, 57 out of 90 is p = 0.99597704
						The prob. of exactly, or more than, 57 out of 90 is p = 0.00743338
						The (one-tailed) prob. of exactly, or greater than, 57 out of 90 is p = 0.007667
	t2	32	58	0,004204	Rej.	The prob. of exactly 58 out of 90 is p = 0.00194041
						The prob. of exactly, or fewer than, 58 out of 90 is p = 0.99791744
						The prob. of exactly, or more than, 58 out of 90 is p = 0.00402296
						The (one-tailed) prob. of exactly, or greater than, 58 out of 90 is p = 0.004204
	t3	54	36	0,977701	Acc.	The prob. of exactly 36 out of 90 is p = 0.013976
						The prob. of exactly, or fewer than, 36 out of 90 is p = 0.03627477
						The prob. of exactly, or more than, 36 out of 90 is p = 0.97770124
						The (one-tailed) prob. of exactly, or fewer than, 36 out of 90 is p = 0.03657
Oracle 04	t1	29	61	0,000542	Rej.	The prob. of exactly 61 out of 90 is p = 0.00026742
						The prob. of exactly, or fewer than, 61 out of 90 is p = 0.99978104
						The prob. of exactly, or more than, 61 out of 90 is p = 0.00048638
						The (one-tailed) prob. of exactly, or greater than, 61 out of 90 is p = 0.000542
	t2	35	55	0,022601	Rej.	The prob. of exactly 55 out of 90 is p = 0.00914793
						The prob. of exactly, or fewer than, 55 out of 90 is p = 0.98684917
						The prob. of exactly, or more than, 55 out of 90 is p = 0.02229876
						The (one-tailed) prob. of exactly, or greater than, 55 out of 90 is p = 0.022601
	t3	51	39	0,914879	Acc.	The prob. of exactly 39 out of 90 is p = 0.03793203
						The prob. of exactly, or fewer than, 39 out of 90 is p = 0.12305322
						The prob. of exactly, or more than, 39 out of 90 is p = 0.9148788
						The (one-tailed) prob. of exactly, or fewer than, 39 out of 90 is p = 0.123126
Oracle 05	t1	26	64	0,000048	Rej.	The prob. of exactly 64 out of 90 is p = 0.00002345
						The prob. of exactly, or fewer than, 64 out of 90 is p = 0.99998517
						The prob. of exactly, or more than, 64 out of 90 is p = 0.00003829
						The (one-tailed) prob. of exactly, or greater than, 64 out of 90 is p = 0.000048
	t2	45	45	0,5	Acc.	The prob. of exactly 45 out of 90 is p = 0.08387112
						The prob. of exactly, or fewer than, 45 out of 90 is p = 0.54193556
						The prob. of exactly, or more than, 45 out of 90 is p = 0.54193556
						The (one-tailed) prob. of exactly, or greater than, 45 out of 90 is p = 0.5
	t3	59	31	0,99897	Acc.	The prob. of exactly 31 out of 90 is p = 0.00105242
						The prob. of exactly, or fewer than, 31 out of 90 is p = 0.00208256
						The prob. of exactly, or more than, 31 out of 90 is p = 0.99896987
						The (one-tailed) prob. of exactly, or fewer than, 31 out of 90 is p = 0.002213

acronyms: **Or.:** Oracle - **T.:** Threshold - **FP/N:** False positives/negatives - **TP/N:** True positives/negatives - **Rej.:** Reject
- **Acc.:** Accept

Source: Elaborated by the author.

probability of success of same event. In the case of this experiment, in particular, each binomial distribution presented in Figure 30 displays the probability of the predominance of FP/N (in red)

and TP/N (in blue). This figure summarizes two main evidences collected from this study: (i) feature-based test oracles can help effectively in testing scenarios to measure quality aspects of TTSs with reduced audio databases; and (ii) variances of threshold strategies are an useful tool to improve the accuracy of the tests, contributing to the reliability of the tests.

Several findings can be collected from an analysis of these numbers:

- *Threshold strategy 3 is useless in all of the cases:* in all of the oracle combinations, the threshold strategy 3 (half of the maximum standard deviation among the feature values) had a poor performance on evaluating the quality of TTSs output produced with reduced audio databases. This is because this strategy makes the threshold too low to identify relevant audio deviations. Then, once the threshold value is low, the test oracle will be more strict. In this study, the test oracles with this strategy have failed almost all of their tests;
- *Threshold strategy 1 performed better than other threshold strategies:* t1 (double of the maximum standard deviation among the feature values) had the better performance in almost all of the oracle strategies. Regardless Oracle 02, t1 achieved positive numbers, with statistical significance, in the rest of the four oracle strategies. Definitely, the numbers collected from t1 show that the oracle combinations can be successfully explored in testing environments;
- *Test oracle with multi-dimensional feature extractors behaves similar or better than test oracles with combinations of single-dimensional feature extractors:* the numbers of our study have revealed that feature-based test oracles composed by multi-dimensional feature extractors can perform similar or even better than test oracles with different single dimensional feature-extractors. This study was conducted with one feature-based test oracle with several feature extractors and four feature-based test oracles with a single multi-dimensional extractor. Through the statistical analysis, it is possible to notice that, in a global context, the better test oracle combination was configured with MFCC feature extractor, which means, a single feature extractor that has multiple dimensions (13). This is a relevant finding due to two reasons: (i) “costs of implementation” – the cost of implementing a single multi-dimensional feature extractor is lower than implementing different feature extractors. Once the effectiveness of the test oracles will be similar for both cases (several single-dimensional feature extractors ext and one multi-dimentional feature extractor), in real-world testing environments, it will be more suitable to explore multi-dimensional feature extractors; (ii) “costs to run tests” – running a bunch of one-dimension feature extractors is costly than running a single multi-dimensional feature extractor. Then, in an analysis regarding time and processing requirements (memory, processors, disk space, etc.), multi-dimensional feature extractors are more recommendable than one-dimensional feature extractors on composing feature-based test oracles.

Figure 30 – Normal distributions of the feature-based test oracle classifications

Source: Elaborated by the author.

Lastly, towards answering RQ1, we highlight that six out of fifteen combinations of feature-based test oracles has shown positive results enough to be integrated in testing environments. As long as statistical analysis have proofed the significance of the number, we can conclude that feature-based test oracles can effectively be explored to test TTS systems with reduced audio database. To do so, the numbers have shown that variations of oracles configurations and threshold strategies can be explored to identify reliable test oracles, depending on the testers' goals.

5.4.5 Threats to Validity

The threats to the validity of this study and the mitigation strategies are analyzed as follows.

Internal validity: the main threat associated with the cause-effect relation between the feature-based test oracles and the quality evaluation of TTSs systems with reduced audio databases is the fact that test oracles always produce assertive and binary outputs (pass/fail). Then, when some novel test oracle approach is being evaluated, researchers have 50% of chance to obtain correct results to each test oracle verdict. This high number can always bias results of some study. To mitigate this chance, we have implemented five test oracles combinations, improving the chances our study to reveal reliable test oracles and raising the confidence of the cause-effect relation between treatments and outcomes.

Likewise, the threshold strategy is an extra internal validity threat. This threat is intimately associated to the feature-based test oracle approach. Then, to mitigate this strategy, this study defined three different threshold strategy to be combined with all of the five configurations of test oracles. Then, due to this, one can claim that this study has a high internal validity.

External validity: two main threats are associated with the generalization of the results observed in this study: (i) four versions of a single real-world TTS systems was explored in our study; and (ii) only the TTS version with female voices, in Brazilian Portuguese, was explored. Consequently, because of these two facts, it can be considered that the major threat that limits the external validity of this study is that it not is able to claim similar results to TTSs implemented to other languages or, even, TTSs with audio databases with male voices. Because of cost issues, it was not possible to implement our own audio database during this research. This brought the necessity of industry partners to create several TTS versions with reduced audio databases. Since this study's collaborator from industry is CPqD, and they had only one huge audio database, this empirical validations were conducted using only CPqD's database, configuring a external threat.

Construct validity: the main construction validity of this study is associated to the possibility of wrong implementation in the test oracles or in the feature extractors. Since the feature extractors used in this study were adapted from previous implementations of audio processing APIs, all of the feature extractors were individually validated and wide adopted by

other researchers, contributing to make the study's construction validity higher. Considering this, it is possible to claim a high validity to the relation between the theory of feature-based test oracles and the experiment's results.

Conclusion validity: one can claim that this study has a high conclusion validity due to three reasons: (i) the experiment's procedure was presented in details, highlighting and justifying all of the decisions; (ii) the results were posted through tables and they are available online to future replications; and (iii) high statistical relevance of the tests ($\alpha = 0.05$) that supports the conclusions.

5.4.6 Findings and Recommendations

This study can be considered as a reference on how to design the approach of feature-based test oracles to support testing activities for a real-world sub-problem of a complex-output system. The main findings collected from this study are: (i) feature-based test oracles can support testing activities to evaluate the quality of TTSs with reduced audio databases; (ii) different configuration of test oracles can be useful to identify more suitable and reliable test oracles; and (iii) the threshold strategy can improve the oracle's effectiveness.

The conduction of this study brought different ideas to future researches. In the future, the replication of this study in other TTSs and with other languages are some of the investigations that should be performed. To do so, it is necessary to establish new cooperation with industry partners.

Finally, it is important to highlight that this experiment resulted in a channel of technological transfer from academia to industry. The technical team of professionals in CPqD are studying the possibility of implementing feature-based test oracles as automatic scripts to analyze the quality of TTSs with small audio database. This process is being supported by the researchers responsible for this experiment.

5.5 On evaluating feature-based test oracles to check artificial prosody problems of TTS systems

The automation of testing activities to identify prosody problems in TTS systems is a non-trivial task. Prosody represents the correct levels of several features such as, stress, rhythm patterns, intonation, and duration of a synthesized speech. In human speeches, prosody brings emphasis, intent, and attitudes to listeners. Also, prosody aspects are essential to make synthesized speeches sound understandable and natural. Prosody imperfections reflect in the perceptual level of the synthesized speech. Then, a TTS that produce speeches with prosody problems will be useless for several applications. Speeches with prosody problems, generally,

are difficult to understand and sometimes “annoying” to users. On the other hand, speeches with good prosody aspects have high intelligibility and naturalness, being soothing to the human ears.

TTS implementations are always dealing with prosody issues because most of the languages do not have any prosodic phrasing in text through phrasal accentuation or punctuation. Then, concatenative TTS should implement their own prosody rules depending on the language. When these rules are poorly implemented, the TTS outputs will sound odd, configuration unnatural voices, and, then, several sentences may be misunderstood.

The evaluation of the quality of prosody aspects in TTS outputs are generally performed through subjective tests. In industry environments, listening tests with human subjects are conducted to evaluate prosody aspects in TTS applications. Generally, a specific MOS score is defined to the prosody aspects of a TTS under evaluation, and when a pre-defined score is reached by that TTS, then it “passes” in the test.

Prosody feature problems are intrinsic in TTS applications, and it is a challenging task to detect their cause in real-world TTSSs. Then, it is hard to figure the cause-effects of prosody problems in TTS, making the automation of testing activities a significant challenge to practitioners. Given this scenario, this experiment simulates prosody problems in TTS outputs to evaluate the applicability of feature-based test oracles in detecting them. The experiment’s objectiveness is to have feature-based test oracle as an object of study to determine the effectiveness on identifying two different prosody problems: (i) rhythm alters; and (ii) pitch modifications.

The perspective of this study is of the tester’s point-of-view and, then, the effect evaluated associated with the perspective of the effectiveness and reliability of feature-based test oracles on dealing with prosody problems in TTS systems. Accordingly, it is observed the number of false verdicts and the number of true verdicts of some feature-based test oracles.

5.5.1 Research Questions and Metrics

The Following Research Questions guide this experiment:

- RQ1 Are feature-based test oracles able to perform practical tests regarding prosody problems?
- RQ2 Among t_1 , t_2 , and t_3 , which is the most suitable threshold strategy to evaluate prosody problems with the same oracle combinations?

The metrics collected to supporting answering RQ1 and RQ2 are the number of false verdicts (FP/N) and true verdicts (TP/N). These two values are the standard metric to evaluate the performance of test oracles.

5.5.2 Subjects

CPqD is the subject explored in this experiment. The language used is Brazilian Portuguese and the speaker gender is female.

5.5.3 Experimental Design and Procedure

Towards raising evidence to answer RQ1 and RQ2, the following experimental procedure was conducted.

As it was stated before, it is necessary to establish it is tough to simulate real prosody problems at the same level in a research environment. To alleviate this, we have explored in this study a research version of CPqD TTS in which users can switch patterns values of pitch and rhythm. This experimental version was created by CPqD team especially to be used in this experiment. This “research” version of CPqD has a friendly GUI from which researchers can synthesize speeches changing the default parameters of the pitch and rhythm. The default values of pitch and rhythm in CPqD TTS is 100. Changing these values to 80, 60, etc. will reflect directly in the prosody of the final synthesized speech. Then, regarding testing activities, it is possible to create “mutated” speeches that are potential candidates to present prosody problems.

Considering this introductory scenario, the following experimental strategy was defined to raise evidence to answer RQ1 and RQ2.

- Considering the 30 sentences explored in previous experiments, it was recorded these 30 sentences with default values to pitch and rhythm. In a feature-based test oracles scenario, these sentences will be used as the reference to our tests;
- Creating six modified group of sentences to simulate prosody problems. Three versions were modified to simulate pitch alternations. Then, using our subject TTS, we synthesized 30 sentences with 80, 60, and 40 as the pitch parameters. In the same way, it was synthesized 30 with 80, 60, and 40 being the rhythm setting, creating a dataset of 180 audios (90 with modified pitch and 90 with modified rhythm). This creation of modified sentences addressed to simulate artificial prosody problems that should be identified by feature-based test oracles;
- After creating a dataset with 180 sentences, we then created five feature-based test oracles with the same configuration presented in our previous experiment (Sec X). The, summarily, one oracle was designed with 17 single-dimension feature extractors (Oracle 01) and each of the other four test oracles were adjusted with a single multi-dimensional feature-based test oracles. Euclidean distance was used in all of the test oracles. Mean was the aggregator factor used in single-multidimensional feature extractors;
- The threshold strategy defined in this study, aiming to raise evidence to answer RQ2, is based on the standard deviation of the values of the feature extractors applied in the

defined dataset. Then, it was defined $t1$ as the double of the maximum standard deviation among the feature results, $t2$ represents the maximum standard deviation calculated from the feature results, and $t3$ is half of $t2$.

After finishing the procedure aforementioned, using the testing environment provided by O-FIm/CO, and taking the pre-defined oracle combinations, we loaded the test for the 30 sentences and their respective modified versions. Through the research version of O-FIm/CO, we collected all of the results from feature extractors and the test oracle verdicts (fails and passes).

A manual analysis on all of the modified versions was carried out. This manual analysis consisted of a subjective listening test to evaluate the TTS's outputs. This analysis is focused on identifying acceptable and unacceptable modified versions of sentences. This analysis provided the following findings:

- **Rhythm:** we noticed this parameter had low effect on the naturalness and intelligibility of the sentences when the parameters 80 and 60 were adjusted. Still, regarding these two modified versions, we noticed only that the sentence pronounced faster than normal. However, the modified versions with 40 as rhythm were extremely affected by regarding intelligibility concepts. In most of the cases, it is not possible to understand the meaning of the sentence; and
- **Pitch:** Similarly to the scenario observed from the rhythm analysis, modified sentences with 80 and 60 had low effect on the intelligibility and prosody problems. In both cases, the reader seems to be reading the sentence with calm. However, when evaluating the sentences created with 40 as the pitch parameter, the speech sounded very unnatural, similarly to “sleepy” person.

Then, given this information, it was possible to establish a template of expected results to calculate false and true verdicts from our test oracle configurations. In this sense, sentences produced with the parameter of 80 and 60 for both, pitch and rhythm, should be accepted (pass) by feature-based test oracles. On the other, sentences modified with the parameter 40 for pitch and rhythm should be rejected (fail).

Table 29 presents the template of verdicts defined to this experiment. Next, we presents the results and discussion about the experiment.

Table 29 – Template with expected verdicts of prosody study

modified version	R - 80	R - 60	R - 40	“I - 80	“I - 60	“I - 40
Expected Oracle Verdict	pass	pass	fail	pass	pass	fail

5.5.4 Experimental Results and Discussion

Table 30 presents the results of the five configurations of feature-based test oracles combined with the three threshold strategies. In the table, one can notice details about the results of the five oracle configurations (in Table 36, O1, O2, O3, O4, and O5) with the three threshold configurations (In Table 36, t1, t2, and t3). Each oracle verdict is displayed in Table 36 as **p** (pass) or **f** (fail). Due to the limit of space, we provided online a full table with the provided verdicts and the results of the features considered by each oracle¹⁴. Table 30 presents each modified version of the 30 sentences are presented in a format to indicate the prosody aspect that was manipulated. For instance, Sentence 01 is shown as follows in the table:

- : “”01S” – original sentence with default values of pitch and rhythm;
- “01-r80” – modified sentence generated with 80 as the rhythm parameter;
- “01-r60” – modified sentence generated with 60 as the rhythm parameter;
- “01-r40” – modified sentence generated with 40 as the rhythm parameter;
- “01-i80” – modified sentence generated with 80 as the pitch parameter;
- “01-i60” – modified sentence generated with 60 as the pitch parameter; and
- “01-i40” – modified sentence generated with 40 as the pitch parameter.

Finally, for each sentence, considering each test oracle, Table 30 presents the Euclidean distance from the reference audio. These values are shown in column “dist” in reference to each row (represented by the sentence). It can be observed that the distance between figured to the reference sentences (01S, 02S, 03S, ... 30S) will always be 0.0, because they have the perfect similarity.

Table 30 – Overall results of feature-based test oracle evaluating prosody problems

Sent.	O1				O2				O3				O4				O5			
	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3
01-i40	0,98	f	f	f	0,45	f	f	f	1,52	f	f	f	0,06	f	f	f	0,54	f	f	f
01-i60	0,64	f	f	f	0,30	p	f	f	0,96	f	f	f	0,03	p	f	f	0,55	p	f	f
01-i80	0,39	p	f	f	0,18	p	p	f	0,61	f	f	f	0,02	p	f	f	0,49	p	f	f
01-r40	0,37	p	f	f	0,10	p	p	p	0,24	p	f	f	0,05	p	f	f	0,46	p	f	f
01-r60	0,22	p	p	f	0,07	p	p	p	0,21	p	p	f	0,03	p	f	f	0,28	p	f	f
01-r80	0,13	p	p	p	0,02	p	p	p	0,15	p	p	f	0,01	p	p	p	0,10	p	p	p
01S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p

Continued on next page

¹⁴ see: [www.labes.icmc.usp.br/\\$\sim\\$rpapes/dissertation.html](http://www.labes.icmc.usp.br/$\sim$rpapes/dissertation.html)

Table 30 continued from previous page

Sent.	O1				O2				O3				O4				O5			
	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3
02-i40	0,99	f	f	f	0,50	f	f	f	1,60	f	f	f	0,06	f	f	f	0,55	f	f	f
02-i60	0,63	f	f	f	0,32	p	f	f	1,02	f	f	f	0,02	p	p	f	0,32	p	f	f
02-i80	0,40	p	f	f	0,20	p	p	f	0,66	f	f	f	0,02	p	p	f	0,42	p	f	f
02-r40	0,50	p	f	f	0,10	p	p	p	0,35	p	f	f	0,04	p	f	f	0,41	p	f	f
02-r60	0,29	p	p	f	0,07	p	p	p	0,20	p	p	f	0,01	p	p	f	0,21	p	p	f
02-r80	0,17	p	p	f	0,03	p	p	p	0,09	p	p	p	0,01	p	p	p	0,14	p	p	f
02S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
03-i40	1,20	f	f	f	0,57	f	f	f	1,89	f	f	f	0,08	f	f	f	0,59	f	f	f
03-i60	0,78	f	f	f	0,38	p	f	f	1,26	f	f	f	0,02	p	f	f	0,44	p	f	f
03-i80	0,50	p	f	f	0,22	p	f	f	0,75	f	f	f	0,02	p	p	f	0,53	p	f	f
03-r40	0,46	p	f	f	0,09	p	p	p	0,18	p	p	f	0,04	p	f	f	0,28	p	f	f
03-r60	0,25	p	p	f	0,04	p	p	p	0,15	p	p	f	0,02	p	p	f	0,15	p	p	f
03-r80	0,09	p	p	p	0,03	p	p	p	0,03	p	p	p	0,01	p	p	p	0,10	p	p	p
03S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
04-i40	1,21	f	f	f	0,58	f	f	f	2,01	f	f	f	0,06	f	f	f	0,70	f	f	f
04-i60	0,83	f	f	f	0,38	p	f	f	1,30	f	f	f	0,03	p	f	f	0,45	p	f	f
04-i80	0,57	p	f	f	0,23	p	f	f	0,84	f	f	f	0,03	p	f	f	0,56	p	f	f
04-r40	0,46	p	f	f	0,10	p	p	p	0,36	p	f	f	0,04	p	f	f	0,26	p	f	f
04-r60	0,29	p	p	f	0,05	p	p	p	0,11	p	p	p	0,03	p	f	f	0,17	p	p	f
04-r80	0,15	p	p	p	0,03	p	p	p	0,14	p	p	f	0,01	p	p	p	0,18	p	p	f
04S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
05-i40	1,17	f	f	f	0,54	f	f	f	1,73	f	f	f	0,08	f	f	f	0,59	f	f	f
05-i60	0,81	f	f	f	0,35	p	f	f	1,21	f	f	f	0,02	p	p	f	0,34	p	f	f
05-i80	0,52	p	f	f	0,21	p	p	f	0,79	f	f	f	0,02	p	f	f	0,36	p	f	f
05-r40	0,35	p	f	f	0,09	p	p	p	0,29	p	f	f	0,05	p	f	f	0,35	p	f	f
05-r60	0,17	p	p	f	0,04	p	p	p	0,16	p	p	f	0,02	p	p	f	0,18	p	p	f
05-r80	0,16	p	p	f	0,02	p	p	p	0,14	p	p	f	0,01	p	p	p	0,10	p	p	p
05S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
06-i40	1,12	f	f	f	0,49	f	f	f	1,74	f	f	f	0,07	f	f	f	0,81	f	f	f
06-i60	0,74	f	f	f	0,33	p	f	f	1,17	f	f	f	0,03	p	f	f	0,68	p	f	f
06-i80	0,50	p	f	f	0,20	p	p	f	0,76	f	f	f	0,03	p	f	f	0,64	p	f	f
06-r40	0,26	p	p	f	0,06	p	p	p	0,17	p	p	f	0,02	p	p	f	0,63	p	f	f
06-r60	0,36	p	f	f	0,04	p	p	p	0,03	p	p	p	0,01	p	p	p	0,27	p	f	f
06-r80	0,12	p	p	p	0,02	p	p	p	0,05	p	p	p	0,01	p	p	f	0,20	p	p	f
06S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
07-i40	1,04	f	f	f	0,49	f	f	f	1,58	f	f	f	0,06	f	f	f	0,51	f	f	f
07-i60	0,66	f	f	f	0,32	p	f	f	1,00	f	f	f	0,03	p	f	f	0,33	p	f	f
07-i80	0,42	p	f	f	0,19	p	p	f	0,67	f	f	f	0,02	p	p	f	0,45	p	f	f
07-r40	0,49	p	f	f	0,10	p	p	p	0,28	p	f	f	0,05	f	f	f	0,36	p	f	f
07-r60	0,30	p	p	f	0,05	p	p	p	0,06	p	p	p	0,02	p	p	f	0,23	p	p	f
07-r80	0,22	p	p	f	0,02	p	p	p	0,03	p	p	p	0,01	p	p	p	0,08	p	p	p
07S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
08-i40	1,17	f	f	f	0,56	f	f	f	1,79	f	f	f	0,06	f	f	f	0,55	f	f	f
08-i60	0,77	f	f	f	0,37	p	f	f	1,17	f	f	f	0,02	p	p	f	0,34	p	f	f
08-i80	0,51	p	f	f	0,22	p	f	f	0,71	f	f	f	0,03	p	f	f	0,45	p	f	f
08-r40	0,40	p	f	f	0,08	p	p	p	0,17	p	p	f	0,05	f	f	f	0,17	p	p	f
08-r60	0,18	p	p	f	0,05	p	p	p	0,07	p	p	p	0,02	p	p	f	0,06	p	p	p
08-r80	0,09	p	p	p	0,02	p	p	p	0,02	p	p	p	0,01	p	p	p	0,07	p	p	p
08S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
09-i40	1,25	f	f	f	0,55	f	f	f	1,94	f	f	f	0,07	f	f	f	0,63	f	f	f
09-i60	0,83	f	f	f	0,36	p	f	f	1,30	f	f	f	0,02	p	p	f	0,38	p	f	f
09-i80	0,50	p	f	f	0,20	p	p	f	0,73	f	f	f	0,02	p	p	f	0,39	p	f	f
09-r40	0,44	p	f	f	0,09	p	p	p	0,55	f	f	f	0,05	p	f	f	0,35	p	f	f
09-r60	0,29	p	p	f	0,04	p	p	p	0,14	p	p	f	0,02	p	p	f	0,12	p	p	p
09-r80	0,08	p	p	p	0,01	p	p	p	0,02	p	p	p	0,01	p	p	p	0,07	p	p	p

Continued on next page

Table 30 continued from previous page

Sent.	O1				O2				O3				O4				O5			
	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3
09S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
10-i40	1,06	f	f	f	0,54	f	f	f	1,72	f	f	f	0,06	f	f	f	0,55	f	f	f
10-i60	0,69	f	f	f	0,36	p	f	f	1,10	f	f	f	0,02	p	p	f	0,43	p	f	f
10-i80	0,44	p	f	f	0,21	p	p	f	0,64	f	f	f	0,03	p	f	f	0,53	p	f	f
10-r40	0,31	p	p	f	0,11	p	p	f	0,24	p	f	f	0,03	p	f	f	0,25	p	p	f
10-r60	0,24	p	p	f	0,07	p	p	p	0,11	p	p	p	0,02	p	p	f	0,11	p	p	p
10-r80	0,09	p	p	p	0,03	p	p	p	0,04	p	p	p	0,01	p	p	p	0,08	p	p	p
10S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
11-i40	1,07	f	f	f	0,53	f	f	f	1,61	f	f	f	0,06	f	f	f	0,58	f	f	f
11-i60	0,71	f	f	f	0,36	p	f	f	1,05	f	f	f	0,02	p	p	f	0,40	p	f	f
11-i80	0,43	p	f	f	0,21	p	p	f	0,61	f	f	f	0,03	p	f	f	0,40	p	f	f
11-r40	0,36	p	f	f	0,12	p	p	f	0,36	p	f	f	0,04	p	f	f	0,31	p	f	f
11-r60	0,18	p	p	f	0,06	p	p	p	0,14	p	p	f	0,02	p	p	f	0,32	p	f	f
11-r80	0,16	p	p	p	0,03	p	p	p	0,17	p	p	f	0,01	p	p	f	0,20	p	p	f
11S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
12-i40	1,05	f	f	f	0,50	f	f	f	1,61	f	f	f	0,07	f	f	f	0,72	f	f	f
12-i60	0,67	f	f	f	0,33	p	f	f	1,04	f	f	f	0,03	p	f	f	0,39	p	f	f
12-i80	0,42	p	f	f	0,20	p	p	f	0,61	f	f	f	0,02	p	f	f	0,47	p	f	f
12-r40	0,44	p	f	f	0,11	p	p	p	0,25	p	f	f	0,05	f	f	f	0,30	p	f	f
12-r60	0,34	p	f	f	0,07	p	p	p	0,17	p	p	f	0,02	p	p	f	0,20	p	p	f
12-r80	0,14	p	p	p	0,03	p	p	p	0,03	p	p	p	0,01	p	p	p	0,16	p	p	f
12S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
13-i40	1,12	f	f	f	0,55	f	f	f	1,79	f	f	f	0,06	f	f	f	0,52	f	f	f
13-i60	0,73	f	f	f	0,36	p	f	f	1,13	f	f	f	0,02	p	p	f	0,40	p	f	f
13-i80	0,45	p	f	f	0,22	p	f	f	0,60	f	f	f	0,02	p	p	f	0,46	p	f	f
13-r40	0,36	p	f	f	0,09	p	p	p	0,25	p	f	f	0,04	p	f	f	0,20	p	p	f
13-r60	0,20	p	p	f	0,05	p	p	p	0,12	p	p	f	0,02	p	p	f	0,19	p	p	f
13-r80	0,12	p	p	p	0,03	p	p	p	0,01	p	p	p	0,01	p	p	p	0,10	p	p	p
13S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
14-i40	1,22	f	f	f	0,59	f	f	f	1,83	f	f	f	0,07	f	f	f	0,74	f	f	f
14-i60	0,84	f	f	f	0,38	p	f	f	1,20	f	f	f	0,04	p	f	f	0,61	p	f	f
14-i80	0,57	p	f	f	0,22	p	f	f	0,75	f	f	f	0,03	p	f	f	0,65	p	f	f
14-r40	0,58	p	f	f	0,12	p	p	f	0,25	p	f	f	0,04	p	f	f	0,45	p	f	f
14-r60	0,34	p	f	f	0,06	p	p	p	0,11	p	p	p	0,02	p	p	f	0,22	p	p	f
14-r80	0,18	p	p	f	0,03	p	p	p	0,01	p	p	p	0,01	p	p	p	0,15	p	p	f
14S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
15-i40	0,98	f	f	f	0,46	f	f	f	1,62	f	f	f	0,06	f	f	f	0,57	f	f	f
15-i60	0,65	f	f	f	0,30	p	f	f	1,04	f	f	f	0,03	p	f	f	0,53	p	f	f
15-i80	0,43	p	f	f	0,19	p	p	f	0,63	f	f	f	0,02	p	p	f	0,51	p	f	f
15-r40	0,22	p	p	f	0,08	p	p	p	0,28	p	f	f	0,03	p	f	f	0,32	p	f	f
15-r60	0,18	p	p	f	0,05	p	p	p	0,15	p	p	f	0,02	p	p	f	0,29	p	f	f
15-r80	0,11	p	p	p	0,02	p	p	p	0,05	p	p	p	0,01	p	p	p	0,11	p	p	p
15S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
16-i40	1,15	f	f	f	0,54	f	f	f	1,85	f	f	f	0,07	f	f	f	0,87	f	f	f
16-i60	0,76	f	f	f	0,35	p	f	f	1,22	f	f	f	0,03	p	f	f	0,58	p	f	f
16-i80	0,51	p	f	f	0,21	p	p	f	0,72	f	f	f	0,03	p	f	f	0,73	p	f	f
16-r40	0,33	p	f	f	0,07	p	p	p	0,23	p	p	f	0,03	p	f	f	0,38	p	f	f
16-r60	0,28	p	p	f	0,03	p	p	p	0,23	p	f	f	0,02	p	p	f	0,41	p	f	f
16-r80	0,15	p	p	p	0,01	p	p	p	0,01	p	p	p	0,01	p	p	p	0,19	p	p	f
16S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
17-i40	1,13	f	f	f	0,56	f	f	f	1,70	f	f	f	0,06	f	f	f	0,56	f	f	f
17-i60	0,74	f	f	f	0,37	p	f	f	1,07	f	f	f	0,02	p	p	f	0,41	p	f	f
17-i80	0,43	p	f	f	0,21	p	p	f	0,58	f	f	f	0,02	p	p	f	0,38	p	f	f
17-r40	0,46	p	f	f	0,10	p	p	p	0,14	p	p	f	0,04	p	f	f	0,18	p	p	f
17-r60	0,27	p	p	f	0,05	p	p	p	0,06	p	p	p	0,02	p	p	f	0,13	p	p	f

Continued on next page

Table 30 continued from previous page

Sent.	O1				O2				O3				O4				O5			
	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3
17-r80	0,11	p	p	p	0,02	p	p	p	0,03	p	p	p	0,01	p	p	f	0,13	p	p	p
17S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
18-i40	1,04	f	f	f	0,51	f	f	f	1,67	f	f	f	0,05	f	f	f	0,51	f	f	f
18-i60	0,66	f	f	f	0,34	p	f	f	1,11	f	f	f	0,02	p	p	f	0,36	p	f	f
18-i80	0,41	p	f	f	0,21	p	p	f	0,68	f	f	f	0,02	p	p	f	0,44	p	f	f
18-r40	0,34	p	f	f	0,10	p	p	p	0,22	p	p	f	0,04	p	f	f	0,42	p	f	f
18-r60	0,24	p	p	f	0,06	p	p	p	0,14	p	p	f	0,02	p	p	f	0,30	p	f	f
18-r80	0,16	p	p	f	0,03	p	p	p	0,04	p	p	p	0,01	p	p	p	0,12	p	p	p
18S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
19-i40	1,17	f	f	f	0,54	f	f	f	1,92	f	f	f	0,07	f	f	f	0,59	f	f	f
19-i60	0,80	f	f	f	0,36	p	f	f	1,30	f	f	f	0,03	p	f	f	0,47	p	f	f
19-i80	0,54	p	f	f	0,21	p	p	f	0,82	f	f	f	0,03	p	f	f	0,55	p	f	f
19-r40	0,40	p	f	f	0,09	p	p	p	0,52	f	f	f	0,04	p	f	f	0,29	p	f	f
19-r60	0,34	p	f	f	0,05	p	p	p	0,16	p	p	f	0,02	p	p	f	0,17	p	p	f
19-r80	0,12	p	p	p	0,02	p	p	p	0,16	p	p	f	0,01	p	p	p	0,07	p	p	p
19S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
20-i40	1,17	f	f	f	0,57	f	f	f	1,79	f	f	f	0,06	f	f	f	0,78	f	f	f
20-i60	0,80	f	f	f	0,38	p	f	f	1,20	f	f	f	0,02	p	p	f	0,49	p	f	f
20-i80	0,54	p	f	f	0,23	p	f	f	0,69	f	f	f	0,02	p	p	f	0,52	p	f	f
20-r40	0,40	p	f	f	0,09	p	p	p	0,15	p	p	f	0,03	p	f	f	0,24	p	p	f
20-r60	0,34	p	f	f	0,06	p	p	p	0,14	p	p	f	0,02	p	p	f	0,13	p	p	f
20-r80	0,12	p	p	p	0,03	p	p	p	0,09	p	p	p	0,01	p	p	p	0,15	p	p	f
20S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
21-i40	1,11	f	f	f	0,56	f	f	f	1,76	f	f	f	0,07	f	f	f	0,56	f	f	f
21-i60	0,72	f	f	f	0,36	p	f	f	1,21	f	f	f	0,02	p	p	f	0,32	p	f	f
21-i80	0,47	p	f	f	0,22	p	f	f	0,75	f	f	f	0,02	p	p	f	0,37	p	f	f
21-r40	0,51	p	f	f	0,11	p	p	f	0,44	p	f	f	0,04	p	f	f	0,43	p	f	f
21-r60	0,21	p	p	f	0,06	p	p	p	0,24	p	f	f	0,02	p	p	f	0,22	p	p	f
21-r80	0,12	p	p	p	0,02	p	p	p	0,17	p	p	f	0,01	p	p	p	0,13	p	p	p
21S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
22-i40	1,06	f	f	f	0,55	f	f	f	1,73	f	f	f	0,06	f	f	f	0,65	f	f	f
22-i60	0,68	f	f	f	0,37	p	f	f	1,10	f	f	f	0,02	p	p	f	0,44	p	f	f
22-i80	0,42	p	f	f	0,22	p	f	f	0,64	f	f	f	0,02	p	p	f	0,50	p	f	f
22-r40	0,27	p	p	f	0,11	p	p	f	0,27	p	f	f	0,03	p	f	f	0,27	p	f	f
22-r60	0,11	p	p	p	0,07	p	p	p	0,08	p	p	p	0,01	p	p	p	0,09	p	p	p
22-r80	0,07	p	p	p	0,03	p	p	p	0,04	p	p	p	0,01	p	p	p	0,07	p	p	p
22S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
23-i40	1,03	f	f	f	0,52	f	f	f	1,65	f	f	f	0,08	f	f	f	0,67	f	f	f
23-i60	0,67	f	f	f	0,34	p	f	f	1,09	f	f	f	0,02	p	p	f	0,35	p	f	f
23-i80	0,43	p	f	f	0,20	p	p	f	0,71	f	f	f	0,02	p	p	f	0,48	p	f	f
23-r40	0,37	p	f	f	0,08	p	p	p	0,17	p	p	f	0,04	p	f	f	0,35	p	f	f
23-r60	0,14	p	p	p	0,05	p	p	p	0,14	p	p	f	0,01	p	p	f	0,22	p	p	f
23-r80	0,10	p	p	p	0,02	p	p	p	0,01	p	p	p	0,01	p	p	p	0,08	p	p	p
23S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
24-i40	0,97	f	f	f	0,51	f	f	f	1,51	f	f	f	0,05	f	f	f	0,66	f	f	f
24-i60	0,62	p	f	f	0,34	p	f	f	1,04	f	f	f	0,02	p	p	f	0,47	p	f	f
24-i80	0,38	p	f	f	0,21	p	p	f	0,58	f	f	f	0,02	p	f	f	0,37	p	f	f
24-r40	0,26	p	p	f	0,10	p	p	p	0,13	p	p	f	0,03	p	f	f	0,29	p	f	f
24-r60	0,18	p	p	f	0,07	p	p	p	0,08	p	p	p	0,02	p	p	f	0,19	p	p	f
24-r80	0,10	p	p	p	0,04	p	p	p	0,05	p	p	p	0,01	p	p	p	0,10	p	p	p
24S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
25-i40	1,18	f	f	f	0,55	f	f	f	1,69	f	f	f	0,07	f	f	f	0,47	f	f	f
25-i60	0,76	f	f	f	0,37	p	f	f	1,09	f	f	f	0,03	p	f	f	0,44	p	f	f
25-i80	0,44	p	f	f	0,21	p	p	f	0,64	f	f	f	0,02	p	p	f	0,44	p	f	f
25-r40	0,30	p	p	f	0,12	p	p	f	0,13	p	p	f	0,05	f	f	f	0,45	p	f	f

Continued on next page

Table 30 continued from previous page

Sent.	O1				O2				O3				O4				O5			
	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3	dist	t1	t2	t3
25-r60	0,20	p	p	f	0,06	p	p	p	0,12	p	p	f	0,02	p	p	f	0,20	p	p	f
25-r80	0,08	p	p	p	0,03	p	p	p	0,01	p	p	p	0,01	p	p	p	0,10	p	p	p
25S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
26-i40	1,02	f	f	f	0,52	f	f	f	1,61	f	f	f	0,06	f	f	f	0,68	f	f	f
26-i60	0,67	f	f	f	0,35	p	f	f	1,02	f	f	f	0,03	p	f	f	0,43	p	f	f
26-i80	0,41	p	f	f	0,20	p	p	f	0,62	f	f	f	0,02	p	p	f	0,46	p	f	f
26-r40	0,60	p	f	f	0,09	p	p	p	0,05	p	p	p	0,07	f	f	f	0,30	p	f	f
26-r60	0,27	p	p	f	0,07	p	p	p	0,12	p	p	f	0,03	p	f	f	0,28	p	f	f
26-r80	0,10	p	p	p	0,04	p	p	p	0,07	p	p	p	0,01	p	p	p	0,11	p	p	p
26S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
27-i40	1,12	f	f	f	0,54	f	f	f	1,74	f	f	f	0,05	f	f	f	0,63	f	f	f
27-i60	0,71	f	f	f	0,36	p	f	f	1,17	f	f	f	0,02	p	p	f	0,50	p	f	f
27-i80	0,46	p	f	f	0,23	p	f	f	0,74	f	f	f	0,02	p	p	f	0,53	p	f	f
27-r40	0,46	p	f	f	0,10	p	p	p	0,26	p	f	f	0,04	p	f	f	0,27	p	f	f
27-r60	0,31	p	p	f	0,07	p	p	p	0,17	p	p	f	0,02	p	p	f	0,20	p	p	f
27-r80	0,11	p	p	p	0,03	p	p	p	0,12	p	p	f	0,01	p	p	p	0,09	p	p	p
27S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
28-i40	1,23	f	f	f	0,59	f	f	f	1,84	f	f	f	0,07	f	f	f	0,68	f	f	f
28-i60	0,80	f	f	f	0,39	p	f	f	1,22	f	f	f	0,03	p	f	f	0,36	p	f	f
28-i80	0,48	p	f	f	0,23	p	f	f	0,73	f	f	f	0,02	p	p	f	0,37	p	f	f
28-r40	0,24	p	p	f	0,10	p	p	p	0,23	p	f	f	0,04	p	f	f	0,38	p	f	f
28-r60	0,16	p	p	p	0,06	p	p	p	0,12	p	p	f	0,02	p	p	f	0,23	p	p	f
28-r80	0,10	p	p	p	0,04	p	p	p	0,08	p	p	p	0,01	p	p	p	0,14	p	p	f
28S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
29-i40	1,20	f	f	f	0,58	f	f	f	1,89	f	f	f	0,08	f	f	f	0,62	f	f	f
29-i60	0,80	f	f	f	0,38	p	f	f	1,29	f	f	f	0,03	p	f	f	0,37	p	f	f
29-i80	0,50	p	f	f	0,23	p	f	f	0,79	f	f	f	0,02	p	f	f	0,43	p	f	f
29-r40	0,52	p	f	f	0,11	p	p	p	0,46	f	f	f	0,03	p	f	f	0,52	p	f	f
29-r60	0,36	p	f	f	0,07	p	p	p	0,21	p	p	f	0,01	p	p	f	0,24	p	p	f
29-r80	0,13	p	p	p	0,03	p	p	p	0,05	p	p	p	0,01	p	p	p	0,08	p	p	p
29S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p
30-i40	1,08	f	f	f	0,55	f	f	f	1,77	f	f	f	0,07	f	f	f	0,49	f	f	f
30-i60	0,71	f	f	f	0,38	p	f	f	1,19	f	f	f	0,02	p	p	f	0,30	p	f	f
30-i80	0,43	p	f	f	0,23	p	f	f	0,71	f	f	f	0,02	p	p	f	0,30	p	f	f
30-r40	0,58	p	f	f	0,12	p	p	f	0,31	p	f	f	0,05	p	f	f	0,45	p	f	f
30-r60	0,28	p	p	f	0,08	p	p	p	0,19	p	p	f	0,02	p	p	f	0,25	p	p	f
30-r80	0,16	p	p	f	0,04	p	p	p	0,14	p	p	f	0,01	p	p	p	0,15	p	p	f
30S	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p	0,00	p	p	p

Source: Elaborated by the author.

Through an overall analysis of Table 30, considering the template of expected results, three aspects can be considered:

- The overall results show that the feature-based test oracles performed well on identifying prosody problems, achieving more true verdicts than false verdicts in all of the five oracle configurations. Rarely the oracle configurations achieved 100% of true verdicts. For instance, the combination of Oracle 04 with t1 has achieved 100% of precision in five occasions (with sentences 7, 8, 12, 25, and 26). On the other hand, some oracle

configurations produced a high number of false verdicts. For example, Oracle 01 with t3 produced several “fail” verdicts, leading to a high number of false verdicts. In this particular case, the low threshold led the oracle to fail several tests;

- Threshold strategy t1 and t2 seemed to be the most effective ones, given the purpose of evaluating artificial prosody imperfections. Analyzing Table 36 it is possible to notice that t1 and t2 performed better than t3. The numbers showed that a threshold between t1 and t2 could improve the oracle effectiveness; and
- Oracle 01, which is composed by 17 single-dimensional feature extractors, has performed better than all of the four other test oracles with multi-dimensional feature extractors. It can be observed that Oracle 01 achieved a high number of true verdicts with t1 and t2. For instance, Oracle 01 with t1 reached 147 of true verdicts out of 180 evaluations. On the other hand, the rest of the test oracles (o2, o3, o4, and o5) had poor performances with t1.

To better answer RQ1, a group of hypotheses was created. This hypothesis should be tested against the numbers presented in Table 30. The hypotheses formulated are:

Null Hypothesis H_0 There is no difference among the numbers of false verdicts and true verdicts collected from feature-based test oracles on evaluating prosody problems.

Alternative Hypothesis H_1 : The number of true verdicts provided by feature-based test oracles is significantly higher than false verdicts.

To test H_0 using the numbers provided in Table 36, it was carried out a binomial test using the number of false verdicts (**FP/N**) and true verdict (**TP/N**). As mentioned in previous experiments, binomial tests are effective instruments to evaluate the effectiveness of novel test oracle approaches. Then, with the support of O-FIm/CO, it was calculated FP/N and TP/N of each oracle configuration combined with the threshold strategies, totaling 15 analysis (o1+t1, o1+t2, o1+t3, o2+t1, o2+t2, o2+t3, o3+t1, o3+t2, o3+t3, o4+t1, o4+t2, o4+t3, o5+t1, o5+t2, and o5+t3).

Table 31 presents the results of the one-tailed binomial tests about the performance of the feature-based test oracles on identifying prosody problems in TTS systems. The tests were adjusted to have a significance level $\alpha < 0,0000001$. The value $< 0,0000001$ is a standard value to statistical tests that have a high number of attempts in binomial tests. Once we had a dataset of 180, statistical specialists defined that this evaluation could be adjusted to this high significance level. Table 31 presents, to each oracle combination, the number of false verdicts (FP/N), the number of true verdicts (TP/N), and the calculated p-value. From these numbers, the table presents what are the test to H_0 regarding each oracle combinations.

Table 31 also presents a comprehensive interpretation with the meaning of each test results. Through this analysis, one can identify all of the probabilities of the numbers of true verdicts to be fewer, greater, or exactly equal to a given TP/N.

Table 31 – Binomial statistical test of feature-based test oracles evaluating prosody problems

Or.	Th.	FP/N	TP/N	p-value	H ₀ ?	meaning
Oracle 01	th 1	33	147	<0.000001	Rej.	The prob. of exactly 147 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 147 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 147 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 147 out of 180 is p < 0.000001
	th 2	48	132	<0.000001	Rej.	The prob. of exactly 132 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 132 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 132 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 132 out of 180 is p < 0.000001
	th 3	72	108	0.004544	Acc.	The prob. of exactly 108 out of 180 is p = 0.00161614
						The prob. of exactly, or fewer than, 108 out of 180 is p = 0.99716586
						The prob. of exactly, or more than, 108 out of 180 is p = 0.00445028
						The (one-tailed) prob. of exactly, or greater than, 108 out of 180 is p = 0.004544
Oracle 02	th 1	68	112	0.000675	Acc.	The prob. of exactly 112 out of 180 is p = 0.0002677
						The prob. of exactly, or fewer than, 112 out of 180 is p = 0.99962469
						The prob. of exactly, or more than, 112 out of 180 is p = 0.00064302
						The (one-tailed) prob. of exactly, or greater than, 112 out of 180 is p = 0.000675
	th 2	48	132	<0.000001	Rej.	The prob. of exactly 132 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 132 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 132 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 132 out of 180 is p < 0.000001
	th 3	60	120	0.000005	Acc.	The prob. of exactly 120 out of 180 is p = 0.00000236
						The prob. of exactly, or fewer than, 120 out of 180 is p = 0.99999778
						The prob. of exactly, or more than, 120 out of 180 is p = 0.00000457
						The (one-tailed) prob. of exactly, or greater than, 120 out of 180 is p = 0.000005
Oracle 03	th 1	65	115	0.00013	Acc.	The prob. of exactly 115 out of 180 is p = 0.00005434
						The prob. of exactly, or fewer than, 115 out of 180 is p = 0.9999348
						The prob. of exactly, or more than, 115 out of 180 is p = 0.00011954
						The (one-tailed) prob. of exactly, or greater than, 115 out of 180 is p = 0.00013
	th 2	48	132	<0.000001	Rej.	The prob. of exactly 132 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 132 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 132 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 132 out of 180 is p < 0.000001
	th 3	66	114	0.00023	Acc.	The prob. of exactly 114 out of 180 is p = 0.00009468
						The prob. of exactly, or fewer than, 114 out of 180 is p = 0.99988046
						The prob. of exactly, or more than, 114 out of 180 is p = 0.00021422
						The (one-tailed) prob. of exactly, or greater than, 114 out of 180 is p = 0.00023
Oracle 04	th 1	62	118	0.000021	Acc.	The prob. of exactly 118 out of 180 is p = 0.00000889
						The prob. of exactly, or fewer than, 118 out of 180 is p = 0.99999079
						The prob. of exactly, or more than, 118 out of 180 is p = 0.0000181
						The (one-tailed) prob. of exactly, or greater than, 118 out of 180 is p = 0.000021
	th 2	36	144	<0.000001	Rej.	The prob. of exactly 144 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 144 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 144 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 144 out of 180 is p < 0.000001
	th 3	69	111	0.001122	Acc.	The prob. of exactly 111 out of 180 is p = 0.00043453
						The prob. of exactly, or fewer than, 111 out of 180 is p = 0.99935698
						The prob. of exactly, or more than, 111 out of 180 is p = 0.00107755
						The (one-tailed) prob. of exactly, or greater than, 111 out of 180 is p = 0.001122
Oracle 05	th 1	68	112	0.000675	Acc.	The prob. of exactly 112 out of 180 is p = 0.0002677
						The prob. of exactly, or fewer than, 112 out of 180 is p = 0.99962469
						The prob. of exactly, or more than, 112 out of 180 is p = 0.00064302
						The (one-tailed) prob. of exactly, or greater than, 112 out of 180 is p = 0.000675
	th 2	48	132	<0.000001	Rej.	The prob. of exactly 132 out of 180 is p < 0.00000001
						The prob. of exactly, or fewer than, 132 out of 180 is p > 0.99999999
						The prob. of exactly, or more than, 132 out of 180 is p < 0.00000001
						The (one-tailed) prob. of exactly, or greater than, 132 out of 180 is p < 0.000001
	th 3	73	107	0.006953	Acc.	The prob. of exactly 107 out of 180 is p = 0.002391
						The prob. of exactly, or fewer than, 107 out of 180 is p = 0.99554972
						The prob. of exactly, or more than, 107 out of 180 is p = 0.00684127
						The (one-tailed) prob. of exactly, or greater than, 107 out of 180 is p = 0.006953

Regarding the hypothesis tests presented, Table 31 reveals that six oracle combination had achieved a high statistical significance, making H₀ to be rejected. These are the case of the following oracle combinations: o1t1, o1t2, o2t2, o3t2, o4t3, and o5t2. Out of 180 evaluations, these test oracles had many true verdicts ranging from 132 to 147. In this context, these six

oracle combinations had a performance that allows them to be explored in testing environments. Regarding the oracle combinations that did not reach statistical significance, the number of TP/N ranged from 107 to 120. Since the statistical significance achieved in this study was high, it was not necessary to present binomial distribution of any oracle configuration.

By the numbers, provided by this study, the oracle configurations showed that in six occasions the number of true verdicts revealed that feature-based test oracles can be effectively explored in testing environments to evaluate prosody problems in TTS applications. This provides a favorable scenario to answer our RQ1, assuring that feature-based test oracles can be useful.

It can be observed that different thresholds can be evaluated to identify better oracle performances. In this particular case of this study, the threshold t2 has achieved a better performance than t1 and t3. Then, through this study's results, one can claim that t2 is the most suitable strategy to support these five configurations of feature-based test oracle on evaluating prosody problems. However, it is necessary to evaluate these numbers aiming at check experimental evidence on the supremacy of t2 in comparisons to t1 and t3.

The difference in the results depending on the threshold analysis is addressed in RQ2 and to better answer this question, a group of hypotheses was created. This hypothesis should be tested against the numbers presented in Table 31. The hypotheses formulated are:

Null Hypothesis H0-rq2: There is no difference among the numbers of false verdicts and true verdicts collected from t2 against t1 and t3.

Alternative Hypothesis H1-rq2: The number of true verdicts provided by t2 is higher t1 and t3.

To test **H0-rq2**, we performed two unpaired t tests:

- t2 versus t1 with a significance level $\alpha = 0.05$; and
- t2 versus t3 with a significance level $\alpha = 0.05$.

The tests performed shown that by conventional criteria, t2 versus t1 (**p-value = 0.0904**) has achieved a difference that is not quite statistically significant. On the other hand, t2 versus t3 (**p-value = 0.0002**) has reached a difference that is considered to be extremely statistically significant.

Finally, providing an answer to RQ2, through the numbers of this study, it is possible to mention that t2 is a better threshold strategy than t3 when using the same oracle configuration to evaluate prosody problems in TTS systems. However, further studies are necessary to proof that t2 performed better than t1 on evaluating the same problem with the same oracle configurations.

5.5.5 Threats to Validity

The threats to the validity of this study can be stated as follows.

Internal validity: this was an exploratory study, and its focus was on evaluating the behavior of feature-based test oracles on detecting prosody problems in TTS systems. Then, it is a narrow-scoped study guided by two RQs. All of the procedure and methodology aspects are presented and justified. In this context, we claim that this study has a high internal validity, contributing to directly association of the outcomes with the treatments.

External validity: since the prosody problems evaluated by the proposed test oracles were artificially generated by hand through a research version of an industrial TTS, the generalization of the results collected through this study has a natural threat. Two threats are highlighted in this context: (i) this study evolves a single TTS with a single voice, language, and gender (Female –Brazilian Portuguese); and (ii) artificial prosody problems are hard to be reproduced. The mitigation strategy we took was considering the most common prosody problems to simulate them correctly. Once prosody problems were directly associated with pitch and rhythm, we performed systematic alterations in this parameters. About the necessity of other TTSs with another language, we did not have access to other research TTSs from industry and then it was impossible to take a mitigation strategy.

Construct validity: the high number of feature extractors explored in this study can be considered a construct validity. As it was mentioned in previous validations, feature-based test oracles have feature extractors as an intrinsic threat, having their quality directly associated with the quality of the feature extractors. To mitigate this problem, we had validated all of the feature extractors individually before exploring them in testing scenarios.

Conclusion validity: studies about test oracles deals with a conclusion threat that is the high chances of having a coincident result achieved during some experiment. Since the possible outputs provided by test oracles are binary (pass/fail), coincidences can promote the dissemination of fake results. Since this study presented results from 15 test oracles combinations, and it was statically proofed that some oracles performed better than others, we claim that this study has a high conclusion validity. Summarily, the threat associated with the high chance of obtaining an FP/N greater than the actual results was mitigated by the evaluation of a large number of oracle combinations.

5.5.6 Final Remarks and overall recommendation

This study has demonstrated how feature-based test oracle can support the implementation of systematic testing environments to TTS systems regarding prosody problems. In general, the test oracles performed well on identifying TTS's outputs with prosody inconsistencies that were artificially generated by hand.

In practical industry environments, the findings of this study can be explored to compare the prosody quality of concatenative systems with the prosody quality of parametric implementations.

Prosody problems are a challengeable issue faced by practitioners. Regularly, when some maintenance is performed in a TTS component, the new version of the TTS will be exposed to prosody problems. Then, a group of trustful feature-based test oracles can be adjusted to perform regression testing activities to evaluate prosody quality aspects after each maintenance. Still, it is important to highlight that because of the difficulty in reproducing real-world prosody problems, it is not assumed that the results of this oracle configurations will be the same in practical scenarios. Regarding this, future studies will be conducted with industry collaborators.

The final recommendations and thought provided through this study are:

- “The approach of feature-based test oracles can be adjusted to work properly to identify prosody problems in particular TTS implementations” – prosody problems can be intimately associated with the internal components of a particular TTS. Different TTS implementations face different prosody problems. This study can be faced as a first-step to practitioners to implement effective testing environments to test their own TTS systems. In this context, previous versions of their TTS application or even similar TTSs can be used as reference when composing a testing scenario: and
- “Threshold alternations can improve the effectiveness of feature-based test oracles” – as it was observed in previous studies, the threshold value is guessed from the testers point-of-view to provide an acceptable limit of the difference between objects under test and the object used as a reference. Then, when the tester has no previous knowledge on feature-based test oracles to a given domain, it is recommendable to provide more than one threshold value and observe the behavior of the test oracles with them. Through this study, it is possible to notice that there is no general recommendation about threshold definitions and their performance will depend on the test purpose. This provided numbers that proof, with statistical significance, that some thresholds perform better than others.

5.6 Final remarks and overview

This section presented five empirical evaluations on the applicability of feature-based test oracles to TTS systems. First, through a pilot study, it was evaluated the usage of feature extractors to speech recognition. Secondly, we have measured feature-based test oracles sensibility through an experiment with multi-language and multi-genders TTSs. Finally, we have explored feature-based test oracles to three real-world challenges faced by practitioners on developing TTS: normalization issues, audio databases sizes, and prosody inconsistencies.

CONCLUSIONS AND FUTURE DIRECTIONS

Systems with sophisticated and complex formats of outputs are becoming pervasive and ubiquitous in contemporary systems. Hypermedia data, audio, three-dimensional environments, and virtual reality are some examples of outputs that complex and non-trivial to the application of software engineering activities. Regardless the output format, verification, validation, and testing are elementary SE activities. Their use raises the chances of market success, and it contributes to the certainty that a given SUT behaves in agreement with its specification. However, the automation of testing activities to systems with complex outputs is challengeable. Conventional testing strategies are not always applicable to these systems., leading practitioners to perform subjective and ad-hoc testing strategies, contributing to the delivery of systems with errors.

In this Ph.D. dissertation, we proposed and evaluated a test oracle approach to support the automation of testing activities to systems with complex outputs – the feature-based approach. The methodology of feature-based test oracles considers extracting features from the SUT’s outputs to use them in the automation of testing activities. To make this approach feasible in practice, we have extended a testing framework called O-Flm/CO, making it possible the automation of test oracles with different configurations. Through an API, O-Flm/CO allows testers to define their test oracles using pre-installed feature extractors and similarity functions.

We have validated feature-based test oracles in the context of TTS systems – a conventional example of systems with complex outputs. Through five empirical studies with real-world TTSs, this dissertation reveals strengths and weakness of feature-based test oracles on supporting testing activities to TTS systems. These empirical studies were focused on real-world sub-problems of TTS systems such as normalization issues, audio databases sizes, and prosody issues. The overall results revealed that feature-based test oracles are a promising testing strategy to systems with complex outputs.

Next section provides a broad discussion about feature-based test oracles (Sec. 6.1), the summary of contributions of this dissertation (Sec. 6.2), future work to be conducted from the

findings provided in through this study (Sec. 6.3), and final remarks (Sec. 6.3).

6.1 Discussion

Several points about feature-based test oracles deserve a broad discussion. Towards raise evidence about the weakness and strengths of the approach proposed in this dissertation, this section presents and discusses the main particular aspects of feature-based test oracles.

6.1.1 *Necessity of specific domain knowledge*

On using the approach of feature-based test oracles, a tester should have at least some basic understanding of feature extraction from SUT's output domain. For instance, the test designer of TTS systems needs to know concepts of audio signal processing to purpose effective test oracles. Also, as soon as this professional is responsible for conducting testing activities, he will be in touch with parameters of feature extractors, results of feature extractors, inconsistencies in similarity functions, a configuration of testing scripts, etc. Then, when some imperfection is detected or, even, when some adjust necessary, this test designer will need for knowledge to take the right decision towards improving the test results.

Still regarding the necessity of previous knowledge on how to deal with some output domain, when conducting the experiments that composed this dissertation, we noticed that minor efforts are necessary when some software framework is available. For instance, on using O-FIm/CO, a tester can explore feature extractors as black-box structures, using only certain parameters to adjust his test oracles. Still considering O-FIm/CO, when the tester has a complete catalog of feature extractors, he will have minor efforts on setting feature-based test oracles. A similar case is when the tester has all of the required feature extractors implemented through a library. However, then there is no feature extractor previously adapted and installed in O-FIm/CO, the tester will need to implement them by himself.

6.1.2 *Feature extractors dependency*

Successful feature-based test oracles depend on well-implemented feature extractors. This dependency, as it was briefly discussed in the experiments (Sec. 5), configures a threat to the feature-based test oracle approach. There is no way to ensure reliability to the oracle results, without a previous validation of the feature extractors to a particular output domain. For instance, if a tester should design a feature-based test oracle for three-dimensional objects produced by a Program **P**, and this tester wants to use a group of feature extractors successfully explored during the test of a Program **R**, then, this tester needs to validate all of these feature extractors with **P**'s outputs. This effort is necessary due to two reasons: (i) must of the feature extractors are structures that work properly in a narrow scope and then it is very common the necessity of adjustments to their proper exploration in other domains; (ii) often, feature extractors provide

an output that is difficult to check manually. Then, it's hard to ensure the correctness of some feature extractors. For instance, a feature extractor that counts the number of bifurcations and a given three-dimensional image of blood vessels. By hand, a tester is not able to check the correctness of a possible feature extractor to this purpose.

The overall recommendation is that validation activities are necessary when feature-based test oracles are evaluating novel domains. This validation processes can be simple, and it can be performed through a group of "test cases" previously evaluated in the context of some features.

6.1.3 *Dependency of an output reference*

Feature-based test oracles depend on an output reference from which specific features are considered as a pattern that should be followed by SUT's outputs. This output reference is, then, assumed to be a standard that SUT's outputs should be similar with. The problem is that testers not always have an output reference that represents all of the desired features from other SUT's outputs.

Typically, the output reference can be defined in several circumstances: through a previous version of the application, a standard pattern of some application, or, even, a similar system implemented in another context (e.g., with other language or under another paradigm). When testers have no reference from which features can be extracted, the feature-based test oracle approach are not applicable. In such cases, testers could then follow other test oracle strategies. For instance, if the tester can define metamorphic relations using the SUT's specification, then approaches of metamorphic-based test oracles can be applied.

6.1.4 *Amortization of costs*

Feature-based test oracles configure a conventional example of the "amortization cost" in SE activities. The concept of amortization of costs is associated with the SE activities in which practitioners or researchers spent a significant amount of time to produce artifacts enough to promote the future reuse of them in several other solutions with much lower efforts.

In the case of feature-based test oracles, the concept of cost amortization is associated with the implementation and validation of feature extractors. Repositories of feature extractors are necessary to disseminate the amortization. The most efficient way to alleviate efforts on using feature-based test oracle approaches in testing activities is through the dissemination of knowledge provided by code repositories. In this way, when a test designer wants to use the feature-based test oracle approach, he can search on these repositories for suitable feature extractors that are already validated to his particular domain. To start this philosophy, all of the audio feature extractors implemented in this study are online available to be downloaded and used in other testing scenarios.

6.1.5 *Feature-based test oracles' limitation*

During the conduction of the empirical validations of this dissertation, some restrictions to feature-based test oracles were noticed:

- The dependency of a framework or API: feature-based test oracles highly need for a previous implemented framework or API to make it possible to evaluate SUT's outputs through the comparison of extracted features. Adjustments of oracle references, threshold setting, report assessments, parameterizations, are some of the required functionalities of a framework that implements the feature-based test oracle approach. For the purpose of this Ph.D. dissertation, we extended the previously implemented testing framework called O-Flm/CO. Then, O-Flm/CO was adjusted to work properly to support the implementation of feature-based test oracles for complex-output systems;
- The dependency of an output reference: as it was previously discussed, the necessity of reference from which features could be extracted limits the application of feature-based test oracles in some output domains;
- Threshold adjustments: to compute threshold values to decide when a given result should be considered correct or incorrect is not a trivial task. This is a characteristic that came from CBIR concepts, and there is no single solution for it. One possible approach to mitigate this problem could be to define ranges of value for the comparison, such as “not accepted – fail”, “suspicious – check”, and “accepted – pass”. This may improve the accuracy of the approach but reduce the degree of automation, since the results in the middle range would require the tester's intervention;
- The necessity of an effective strategy to collect SUT's outputs: to provide a high level of automation, it is necessary to implement automated strategy to obtain SUT's outputs to be tested. This is necessary because feature-based test oracles deal with SUT's outputs, evaluating a bunch of features and taking the features of a reference as a pattern. To implement effective feature-based test oracles, besides the necessity of references, one could state that an effective strategy to collect SUTs output is required. In the context of this study, all of the datasets were created by hand without an explicit or automated method, making the data collection a time-consuming activity. Under this perspective, the need for an automated and systematic strategy to collect several SUT's outputs configures a limitation to the approach. Since some considerable efforts should be spent on implementing effective strategies to collect data, the industrial applicability of the feature-based test oracles can be impaired; and
- Difficulties on checking the correctness of the test oracles' verdicts: once there are no expected results to SUT's outputs being evaluated by feature-based test oracles, testers often should check the correctness of oracle verdicts by hand. This activity is most

common when a novel domain is being tested in the context of the oracle approach proposed in this thesis. To mitigate this limitation, in O-FIm/CO, there is a research mode that provides detailed reports about the test executions with all of the results and other relevant information about the test load.

6.2 Summary of contributions

This section revisits the contributions mentioned in the introduction section of this dissertation. The following contributions, to be considered at different levels, were formed through this dissertation:

- The approximation of an SE method to complex-output domains through the approach called feature-based test oracles;
- The establishment of research that drives other approaches on how to deal with unconventional and sophisticated domains that limit the application of SE activities;
- The proposal of feature-based test oracles, a sub-category of specification-based test oracles, to promote the automation of testing activities in several complex-output domains. In particular, this contribution is considered the most relevant one, since the approach can be extended to several other domains;
- A systematic mapping about test strategies and quality evaluation of TTS systems;
- A catalog of audio feature extractors adapted to their use in testing scenarios;
- A software framework that implements feature-based test oracles to several complex-output domains;
- A broad taxonomy on test oracles that can be explored by other researchers as a starting point to novel test oracle approaches;
- Taking TTS applications as the major target, this dissertation provides contributions on showing a step-by-step methodology to apply feature-based test oracles in particular domains;
- A broad presentation of the problems associated with complex-output systems can be considered a contribution, once it can instigate other researchers to dedicate efforts to provide automated software engineering processes to them; and
- Regarding the software testing research field, approaches similar to the feature-based test oracles can lead software testers to think outside the box when designing and conducting systematic testing strategies for non-conventional SUT's output domains.

6.3 Future work

This dissertation enabled the identification of several future directions to the feature-based approach. In this section, we discuss future research efforts in short, medium, and long terms.

In short terms, besides some initiatives to improve the results of our experiments, we will conduct studies with a new complex-output domain – three-dimensional medical objects. Then, O-FIm/CO will be explored with a group of feature extractors for three-dimensional objects. Besides, we will implement new functionalities in the research version of O-FIm/CO. Using the results of the experiments presented, we will perform a linear scope analysis to identify the most sensible feature extractors to a given TTS sub-problem. Finally, in cooperation with industry partners, we will conduct an MOS evaluation to the same test purposes evaluated in our experiments towards raise evidence about trade-offs on using feature-based test oracles instead of MOS to quality evaluation of TTS systems.

In middle terms, we will be in touch with CPqD team to apply some of the feature-based test oracles in real-world industry environments. Through this cooperation, we will measure the efficacy of feature-based test oracles applied in practice by practitioners. Likewise, we expect the identification of other sub-problems in which feature-based test oracles are applicable. Through this cooperation, we will define other experiments towards revealing strengths and weakness of our approach.

In long terms, we will concentrate some research efforts to integrate the software framework O-FIm/CO with testing automation environments such as JUnit and Selenium. The integration with might allows the development of testing strategies that validates complex-output web applications, which is a problem that has recently been explored in industry. O-FIm/CO's integration with some mutation testing tools is also in our plans. However, this integration should be done in a way to affect specific-output domains, since major of the mutation testing tools interact with the user through the standard output, as plain texts (Strings). These future efforts are presented as long-term studies because the idea is to recruit undergraduate students and master's students to implement all of them, promoting the creating of human resources through this dissertation.

6.4 Final remarks

The automation of testing activities to complex-output systems is the broad goal that can not be settled in the standard fashion. Research initiatives are necessary to approach academia and industry.

In this dissertation, we have proposed a novel test oracle approach called feature-based test oracles to support the automation of testing activities in complex-output systems. Then, to

support our approach, we have extended a testing framework called O-FIm/CO to meet the concepts of feature-based test oracles. After that, to provide focus, we have limited our validations and empirical analysis to TTS systems. Despite that, the general concept of feature-based test oracles can be stated and applied in several other complex-output domains to automate testing activities. For example, processed images, web applications, three-dimensional environments, virtual reality applications, and others. Finally, it is important to highlight that several artifacts provided by this study are being evaluated to be applied in practice by CPqD to automate testing activities of TTS systems.

It is expected that the studies of this dissertation instigate other researchers to dedicate efforts towards approaching SE and complex-output systems. This is necessary because SE practices need to follow the technological evolution of the industry. Further, the mainstream adoption of technologies and the market competitiveness of the software industry lead to the delivery of software poorly tested and with low quality. In addition to that, the complexity that is natural for most of the contemporary applications severely limits the use of testing activities. The SE research community is, then, with the responsibility of presenting alternative and flexible approaches to make SE Activities in consonance with the current market and society demands. This dissertation provided alternatives to deal with complex-output systems through a novel test oracle approach, enabling possibilities of technology transfer.

BIBLIOGRAPHY

AFSHAN, S.; MCMINN, P.; STEVENSON, M. Evolving readable string test inputs using a natural language model to reduce human oracle cost. In: **Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)**. Luxembourg: [s.n.], 2013. p. 352–361. Cited on page 45.

AGARWAL, D.; TAMIR, D.; LAST, M.; KANDEL, A. A comparative study of artificial neural networks and info-fuzzy networks as automated oracles in software testing. **IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans**, v. 42, n. 5, p. 1183–1193, 2012. Cited 2 times on pages 48 and 60.

AGGARWAL, K.; SINGH, Y.; KAUR, A.; SANGWAN, O. A neural net based approach to test oracle. **ACM SIGSOFT Software Engineering Notes**, v. 29, n. 3, p. 1–6, 2004. Cited 2 times on pages 60 and 61.

AICHERNIG, B.; GRIESMAYER, A.; JOHNSEN, E.; SCHLATTE, R.; STAM, A. Conformance testing of distributed concurrent systems with executable designs. In: **Proceedings of the 7th International Symposium on Formal Methods for Components and Objects (FMCO 2009)**. Sophia Antipolis, France: [s.n.], 2009. p. 61–81. Cited on page 45.

ALBADRI, A. H. Interactive simulation in teaching science labs (reality and expectations). In: **2015 Fifth International Conference on e-Learning (econf)**. [S.l.: s.n.], 2015. p. 458–464. Cited on page 68.

ALIAS, F.; SEVILLANO, X.; SOCORO, J.; GONZALVO, X. Towards high-quality next-generation text-to-speech synthesis: A multidomain approach by automatic domain classification. **Audio, Speech, and Language Processing, IEEE Transactions on**, v. 16, n. 7, p. 1340–1354, sept. 2008. ISSN 1558-7916. Cited on page 73.

ALMOG, D.; HEART, T. Developing the basic verification action (BVA) structure towards test oracle automation. In: **Proceedings of the International Conference on Computational Intelligence and Software Engineering (CiSE 2010)**. Wuhan, China: [s.n.], 2010. p. 1–4. Cited on page 60.

ALROUQI, H.; ALARIFI, A.; ALNAFESSAH, A.; ALHADHRAMI, S.; AL-KHALIFA, H. S.; AL-SALMAN, A. S.; AL-AMMAR, M. A. Evaluating arabic text-to-speech synthesizers for mobile phones. In: **Digital Information Management (ICDIM), 2015 Tenth International Conference on**. [S.l.: s.n.], 2015. p. 89–94. Cited on page 31.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. 1. ed. New York, NY, USA: Cambridge University Press, 2008. ISBN 0521880386. Cited 2 times on pages 27 and 35.

_____. _____. first. Cambridge, UK: Cambridge University Press, 2008. Cited 3 times on pages 36, 37, and 42.

ANDREWS, J.; ZHANG, Y. General test result checking with log file analysis. **IEEE Transactions on Software Engineering**, v. 29, n. 7, p. 634–648, 2003. Cited 3 times on pages 45, 57, and 58.

ANGUERA, X.; PEREZ, N.; URRUELA, A.; OLIVER, N. Automatic synchronization of electronic and audio books via tts alignment and silence filtering. In: **2011 IEEE International Conference on Multimedia and Expo**. [S.l.: s.n.], 2011. p. 1–6. ISSN 1945-7871. Cited on page 31.

ASRAFI, M.; LIU, H.; KUO, F.-C. On testing effectiveness of metamorphic relations: A case study. In: **Proceedings of the 5th International Conference on Secure Software Integration and Reliability Improvement (SSIRI 2011)**. Jeju Island, Korea: [s.n.], 2011. p. 147–156. Cited on page 44.

BACH, B.; RICHE, N. H.; HURTER, C.; MARRIOTT, K.; DWYER, T. Towards unambiguous edge bundling: Investigating confluent drawings for network visualization. **IEEE Transactions on Visualization and Computer Graphics**, v. 23, n. 1, p. 541–550, Jan 2017. ISSN 1077-2626. Cited on page 68.

BARESI, L.; YOUNG, M. **Test Oracles**. Eugene, OR, U.S.A., 2001. <<http://bit.ly/9fkDwJ>> - Access: 09/02/2011. Cited 7 times on pages 36, 41, 42, 48, 53, 54, and 93.

BARKHODA, W.; ZAHIRAZAMI, B.; BAHRAMPOUR, A.; SHAHRYARI, O.-K. A comparison between allophone, syllable, and diphone based TTS systems for Kurdish language. In: **Proceedings of the 9th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT 2009)**. Ajman, UAE: [s.n.], 2009. p. 557–562. Cited on page 88.

BARR, E.; HARMAN, M.; MCMINN, P.; SHAHBAZ, M.; YOO, S. The oracle problem in software testing: A survey. **IEEE Transactions on Software Engineering**, v. 41, n. 5, p. 507–525, May 2015. Cited 3 times on pages 28, 51, and 53.

BEIZER, B. **Software Testing Techniques**. 2nd. ed. New York, NY, USA: International Thomson Computer Press, 1990. 580 p. ISBN 0-442-20672-0. Cited 2 times on pages 48 and 50.

BERGAMASCO, L. C. C.; OLIVEIRA, R. A. P.; WECHSLER, H.; DAJUDA, C.; DELAMARO, M.; NUNES, F. L. S. Content-based image retrieval of 3d cardiac models to aid the diagnosis of congestive heart failure by using spectral clustering. In: **2015 IEEE 28th International Symposium on Computer-Based Medical Systems**. [S.l.: s.n.], 2015. p. 183–186. ISSN 1063-7125. Cited on page 30.

BERTOLINO, A. Software testing research and practice. In: BORGER, E.; GARGANTINI, A.; RICCOBENE, E. (Ed.). **Abstract State Machines 2003**. [S.l.]: Springer Berlin Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2589). p. 1–21. Cited 2 times on pages 36 and 37.

_____. Software testing research: Achievements, challenges, dreams. In: **Future of Software Engineering, 2007. FOSE '07**. [S.l.: s.n.], 2007. p. 85–103. Cited 4 times on pages 27, 29, 68, and 69.

_____. _____. In: **Proceedings of the Future of Software Engineering (FOSE 2007)**. Minneapolis, MN, USA: [s.n.], 2007. p. 85–103. Cited 3 times on pages 36, 37, and 39.

BIEMAN, J.; YIN, H. Designing for software testability using automated oracles. In: **Proceedings of the International Test Conference (ITC 1992)**. Baltimore, MD: [s.n.], 1992. p. 900–907. Cited on page [42](#).

BRESOLIN, A. A. **Estudo do Reconhecimento de Voz para o Acionamento de Equipamentos Elétricos via Comandos em Português**. Dissertação (Master Thesis) — Centro de Ciências Tecnológicas, Universidade do Estado de Santa Catarina, Florianópolis, SC, 2003. Cited on page [108](#).

BRIAND, L.; LABICHE, Y. A UML-based approach to system testing. **Software and Systems Modeling**, v. 1, n. 1, p. 10–42, 2002. Cited on page [56](#).

BRIAND, L.; LABICHE, Y.; SUN, H. Investigating the use of analysis contracts to improve the testability of object-oriented code. **Software - Practice & Experience**, v. 33, n. 7, p. 637–672, 2003. Cited on page [56](#).

BUENO, J. M.; CHINO, F. J. T.; TRAINA, A. J. M.; JR., C. T.; MARQUES, P. M. A. How to add content-based image retrieval capability in a pacs. In: **Proceedings of the 15th IEEE International Conference on Computer Based Medical Systems (CBMS 2002)**. Maribor, Slovenia: [s.n.], 2002. p. 321–326. Cited 2 times on pages [30](#) and [90](#).

CAFEO, B.; MASIERO, P. Contextual integration testing of object-oriented and aspect-oriented programs: A structural approach for Java and AspectJ. In: **Proceedings of the 25th Brazilian Symposium on Software Engineering (SBES 2011)**. Sao Paulo, Brazil: [s.n.], 2011. p. 214–223. Cited on page [36](#).

CALLOU, D.; LEITE, Y. **Iniciação à fonética e à fonologia**. fifth. Rio de Janeiro/RJ: Zahar, 1995. v. 1. Cited on page [108](#).

CASTRO, J. R.; MALDONADO, J. A.; GARCIA, A. A.; CASTANON-PUGA, M.; FIGUEROA, E. Towards an intelligent decision-making support in health information systems: Lightweight hybrid data mining class library for object-oriented applications. In: **2016 SAI Computing Conference (SAI)**. [S.l.: s.n.], 2016. p. 464–468. Cited on page [68](#).

CHAN, W.; HO, J.; TSE, T. Piping classification to metamorphic testing: An empirical study towards better effectiveness for the identification of failures in mesh simplification programs. In: **Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)**. Beijing, China: [s.n.], 2007. v. 1, p. 397–404. Cited on page [43](#).

CHAN, W.; TSE, T. Oracles are hardly attain'd, and hardly understood: Confessions of software testing researchers. In: **Proceedings of the 13rd International Conference on Quality Software (QSIC 2013)**. Boston, MA, USA: [s.n.], 2013. p. 245–252. Cited 3 times on pages [44](#), [46](#), and [47](#).

CHAPPELL, D. T.; HANSEN, J. H. A comparison of spectral smoothing methods for segment concatenation based speech synthesis. **Speech Communication**, v. 36, n. 3–4, p. 343–373, 2002. ISSN 0167-6393. Cited on page [88](#).

CHEN, H. Y.; TSE, T. Equality to equals and unequals: A revisit of the equivalence and nonequivalence criteria in class-level testing of object-oriented software. **IEEE Transactions on Software Engineering**, v. 39, n. 11, p. 1549–1563, 2013. Cited on page [36](#).

CHEN, J.; LIN, M.; YU, K.; SHAO, B. When a GUI regression test failed, what should be blamed? In: **Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)**. Montreal, Canada: [s.n.], 2012. p. 467–470. Cited on page 46.

CHEN, J.; SUBRAMANIAM, S. Specification-based testing for GUI-based applications. **Software Quality Journal**, v. 10, n. 3, p. 205–224, 2002. Cited on page 45.

CHEN, T.; FENG, J.; TSE, T. Metamorphic testing of programs on partial differential equations: a case study. In: **Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC 2002)**. Oxford, England: [s.n.], 2002. p. 327–333. Cited on page 60.

CHEN, T.; HO, J.; LIU, H.; XIE, X. An innovative approach for testing bioinformatics programs using metamorphic testing. **BMC Bioinformatics**, BioMed Central, v. 10, n. 1, p. 1–12, 2009. Cited on page 60.

CHEN, T.; TSE, T.; ZHOU, Z. Fault-based testing without the need of oracles. **Information and Software Technology**, v. 45, n. 1, p. 1–9, 2003. Cited on page 59.

CHEN, T. Y.; CHEUNG, S. C.; YIU, S. M. **Metamorphic Testing: a New Approach for Fenerating Next Test Cases**. [S.l.], 1998. <<http://bit.ly/1nTs6HM>> - Access: 02/13/2014. Cited on page 59.

CHEON, Y.; AVILA, C. Automating java program testing using OCL and AspectJ. In: **Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG 2010)**. Las Vegas, NV, USA: [s.n.], 2010. p. 1020–1025. Cited on page 56.

CHEON, Y.; LEAVENS, G. A simple and practical approach to unit testing: The JML and JUnit way. In: **Proceedings of the European Conference on Object-Oriented Programming (ECOOP 2002)**. Malaga, Spain: [s.n.], 2002. p. 1789–1901. Cited on page 54.

CHEONG, C. Y.; LI, M. H.; GOH, R. S. M. Genomedesign: A bioinformatics program for genome and multiplex gene synthesis. In: **2011 5th International Conference on Bioinformatics and Biomedical Engineering**. [S.l.: s.n.], 2011. p. 1–4. ISSN 2151-7614. Cited on page 68.

CONTOLINI, M.; STOLMENOV, K.; JUNQUA, J. C. Voice technologies for telephony services. In: **Consumer Communications and Networking Conference, 2004. CCNC 2004. First IEEE**. [S.l.: s.n.], 2004. p. 662–664. Cited on page 31.

COUTINHO, V. A.; OLIVEIRA, H. Um sistema para classificação automática de gêneros musicais. In: UFPE. **Proceedings of the XX Congresso de Iniciação Científica da Universidade Federal de Pernambuco (UFPE)**. Recife/PE, 2012. p. 1 – 10. Cited on page 110.

CRAVEN, M. P.; FABRICATORE, C. Game features of cognitive training. In: **2016 International Conference on Interactive Technologies and Games (ITAG)**. [S.l.: s.n.], 2016. p. 42–49. Cited on page 68.

CROCKER, D. Perfect developer: A tool for object-oriented formal specification and refinement. In: **Proceedings of the International Symposium of Formal Methods Europe (FME 2003) – Tools Exhibition Notes**. Pisa, Italy: [s.n.], 2003. p. 1–5. Cited on page 57.

DAHIYA, S. A statistical block fading channel model for multiuser massive mimo system. In: **2016 International Conference on Signal Processing and Communications (SPCOM)**. [S.l.: s.n.], 2016. p. 1–5. Cited on page [68](#).

DAI, G.; BAI, X.; WANG, Y.; DAI, F. Contract-based testing for Web services. In: **Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)**. Beijing, China: [s.n.], 2007. v. 1, p. 517–526. Cited on page [55](#).

DATTA, R.; JOSHI, D.; LI, J.; WANG, J. Z. Image Retrieval: Ideas, Influences, and Trends of the New Age. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 40, n. 2, p. 1–60, april 2008. ISSN 0360-0300. Cited on page [90](#).

DAVIS, M. D.; WEYUKER, E. J. Pseudo-oracles for non-testable programs. In: **Proceedings of the ACM conference (ACM 1981)**. New York, NY, USA: [s.n.], 1981. p. 254–257. Cited 5 times on pages [41](#), [46](#), [47](#), [48](#), and [49](#).

DAY, J.; GANNON, J. A test oracle based on formal specifications. In: **Proceedings of the 2nd Conference on Software Development Tools, Techniques, and Alternatives (SDTTA 1985)**. Los Alamitos, CA, USA: [s.n.], 1985. p. 126–30. Cited on page [41](#).

DELAMARO, M.; NUNES, F.; OLIVEIRA, R. Using concepts of content-based image retrieval to implement graphical testing oracles. **Software Testing, Verification and Reliability**, v. 23, n. 3, p. 171–198, 2013. Cited 4 times on pages [43](#), [45](#), [47](#), and [98](#).

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. 1st. ed. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda, 2007. 394 p. (Campus). ISBN 9788535226348. Cited on page [27](#).

_____. _____. 2st. ed. Rio de Janeiro, RJ, Brasil: Elsevier/Câmpus Editora Ltda, 2016. 400 p. (Campus). ISBN 9788535283525. Cited 2 times on pages [29](#) and [37](#).

DELAMARO, M. E.; NUNES, F. Lourdes dos S.; OLIVEIRA, R. A. P. Using concepts of content-based image retrieval to implement graphical testing oracles. **Software Testing, Verification and Reliability**, v. 23, n. 3, p. 171–198, 2013. ISSN 1099-1689. Cited 2 times on pages [30](#) and [90](#).

DING, J.; WU, T.; WU, D.; LU, J.; HU, X. Metamorphic testing of a Monte Carlo modeling program. In: **Proceedings of the 6th International Workshop on Automation of Software Test (AST 2011)**. Waikiki, Honolulu, HI, USA: [s.n.], 2011. p. 1–7. Cited on page [60](#).

DOBRISEK, S.; GROS, J.; MIHELIC, F.; PAVESIC, N. Homer: a voice-driven text-to-speech system for the blind. In: **Industrial Electronics, 1999. ISIE '99. Proceedings of the IEEE International Symposium on**. [S.l.: s.n.], 1999. v. 1, p. 205–208 vol.1. Cited on page [31](#).

D'SOUZA, D.; GOPINATHAN, M. Computing complete test graphs for hierarchical systems. In: **Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006)**. Pune, India: [s.n.], 2006. p. 70–79. Cited on page [45](#).

DURELLI, V.; ARAUJO, R.; SILVA, M.; OLIVEIRA, R.; MALDONADO, J.; DELAMARO, M. A scoping study on the 25 years of research into software testing in Brazil and an outlook on the future of the area. **Journal of Systems and Software**, v. 86, n. 4, p. 934–950, 2013. Cited on page [43](#).

DURELLI, V. H. S.; ENDO, A. T.; SIMAO, A.; DELAMARO, M. E. Towards envisaging software testing in a pervasive computing world. *IEEE Computer Society, Los Alamitos, CA, USA*, v. 0, p. 201–205, 2012. Cited on page [28](#).

DURRIEU, G.; WAESELYNCK, H.; WIELS, V. LETO – a lustre-based test oracle for airbus critical systems. In: **Proceedings of the 13th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2009)**. Eindhoven, The Netherlands: [s.n.], 2009. p. 7–22. Cited on page [53](#).

DUSTIN, E.; GARRETT, T.; GAUF, B. **Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality**. first. Boston, USA: Addison-Wesley Professional, 2009. Cited on page [38](#).

EBERT, C.; KUHRMANN, M.; PRIKLADNICKI, R. Global software engineering: Evolution and trends. In: **2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)**. [S.l.: s.n.], 2016. p. 144–153. ISSN 2329–6313. Cited on page [27](#).

ECKHARDT D.E., J.; LEE, L. D. A theoretical basis for the analysis of multiversion software subject to coincident errors. *IEEE Transactions on Software Engineering*, SE-11, n. 12, p. 1511–1517, 1985. Cited on page [61](#).

EDWARDS, S. A framework for practical, automated black-box testing of component-based software. *Software Testing Verification and Reliability*, v. 11, n. 2, p. 97–111, 2001. Cited on page [55](#).

FELDER, M.; MORZENTI, A. Validating real-time systems by history-checking TRIO specifications. In: **Proceedings of the International Conference on Software Engineering (ICSE 1992)**. Melbourne, Australia: [s.n.], 1992. p. 199–211. Cited on page [58](#).

FERRARI, F.; MALDONADO, J.; RASHID, A. Mutation testing for aspect-oriented programs. In: **Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST 2008)**. Lillehammer, Norway: [s.n.], 2008. p. 52–61. Cited on page [36](#).

FUJINAGA, I. **Adaptive Optical Music Recognition**. Tese (Ph.D. Thesis) — McGill University, Montreal, QC, Canada, 1997. Cited on page [105](#).

GERAZOV, B.; SHUTINOSKI, G.; ARSOV, G. A novel quasi-diphone inventory approach to text-to-speech synthesis. In: **MELECON 2008 - The 14th IEEE Mediterranean Electrotechnical Conference**. [S.l.: s.n.], 2008. p. 799–804. Cited on page [73](#).

GIANNAKOPOULOU, D.; BUSHNELL, D.; SCHUMANN, J.; ERZBERGER, H.; HEERE, K. Formal testing for separation assurance. *Annals of Mathematics and Artificial Intelligence*, v. 63, p. 5–30, 2011. Cited on page [45](#).

GOPI, A.; P, S. D.; T, S.; K, B. V. Implementation of malayalam text to speech using concatenative based tts for android platform. In: **Control Communication and Computing (ICCC), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 184–189. Cited on page [31](#).

GUDERLEI, R.; JUST, R.; SCHNECKENBURGER, C.; SCHWEIGGERT, F. Benchmarking testing strategies with tools from mutation analysis. In: **Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW 2008)**. Berlin, Germany: [s.n.], 2008. p. 360–364. Cited on page [45](#).

GUDIVADA, V. N.; RAGHAVAN, V. V. Content-Based Image Retrieval Systems. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 28, n. 9, p. 18–22, september 1995. ISSN 0018-9162. Cited on page 30.

GUPTA, R.; BANVILLE, H. J.; FALK, T. H. Multimodal physiological quality-of-experience assessment of text-to-speech systems. **IEEE Journal of Selected Topics in Signal Processing**, PP, n. 99, p. 1–1, 2016. ISSN 1932-4553. Cited on page 68.

HAGAR, J.; BIEMAN, J. Using formal specifications as test oracles for system-critical software. **ACM SIGAda Ada Letters**, v. 16, n. 6, p. 55–72, 1996. Cited on page 54.

HAMLET, R. G. Software quality, software process, and software testing. **Advances in Computers**, v. 41, p. 191–229, 1995. Cited 2 times on pages 42 and 47.

HARMAN, M.; JIA, Y.; ZHANG, Y. Achievements, open problems and challenges for search based software testing. In: **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.: s.n.], 2015. p. 1–12. ISSN 2159-4848. Cited on page 27.

HARMAN, M.; KIM, S.; LAKHOTIA, K.; MCMINN, P.; YOO, S. Optimizing for the number of tests generated in search based test data generation with an application to the oracle cost problem. In: **Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops (ICSTW 2010)**. Paris, France: [s.n.], 2010. p. 182–191. Cited on page 55.

HARMAN, M.; MCMINN, P.; SHAHBAZ, M.; YOO, S. **A Comprehensive Survey of Trends in Oracles for Software Testing**. [S.l.], 2013. <<http://bit.ly/1kFX0T3>> - Access: 12/04/2013. Cited 9 times on pages 28, 39, 42, 44, 48, 51, 53, 60, and 61.

HOFFMAN, D. A taxonomy for test oracles. In: **Proceedings of the 11th International Quality Week (QW 98)**. San Francisco, CA, USA: [s.n.], 1998. p. 1–8. Cited 3 times on pages 42, 48, and 51.

_____. Using oracles in test automation. In: **Proceedings of the 19th Pacific Northwest Software Quality Conference (PNSQC 2001)**. Portland, OR, USA: [s.n.], 2001. p. 91–102. Cited 5 times on pages 38, 40, 42, 48, and 51.

_____. **Using Oracles in Testing and Test Automation**. 2006. LogiGear – Software Testing. Disponível em: <<http://goo.gl/d3Fb6>>. Cited 5 times on pages 28, 39, 40, 48, and 51.

HOFFMAN, D.; STROOPER, P. Automated module testing in prolog. **IEEE Transactions on Software Engineering**, v. 17, n. 9, p. 934–943, 1991. Cited on page 45.

HOOK, D.; KELLY, D. Testing for trustworthiness in scientific software. In: **Proceedings of the ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE 2009)**. Vancouver, Canada: [s.n.], 2009. p. 59–64. Cited on page 45.

HOWDEN, W. Reliability of the path analysis testing strategy. **IEEE Transactions on Software Engineering**, SE–2, n. 3, p. 208–215, 1976. Cited on page 48.

_____. Theoretical and empirical studies of program testing. **IEEE Transactions on Software Engineering**, SE–4, n. 4, p. 293–298, 1978. Cited 2 times on pages 41 and 47.

HUMBLE, J.; FARLEY, D. **Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation**. first. Boston, USA: Addison-Wesley Professional, 2010. Cited on page 39.

HUMMEL, O.; ATKINSON, C. Automated harvesting of test oracles for reliability testing. In: **Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)**. Edinburgh, UK: [s.n.], 2005. v. 2, p. 196–202. Cited on page 61.

HUNTER, C.; STROOPER, P. Systematically deriving partial oracles for testing concurrent programs. In: **Proceedings of the 24th Australasian Computer Science Conference (ACSC 2001)**. Gold Coast, Australia: [s.n.], 2001. p. 83–91. Cited 2 times on pages 36 and 50.

IEEE. Systems and software engineering – vocabulary. **ISO/IEC/IEEE 24765:2010(E)**, v. 1, p. 1–418, 15 2010. Cited 4 times on pages 35, 53, 54, and 56.

IMPERATORE, P.; SALVADORI, E.; CHLAMTAC, I. Path loss measurements at 3.5 GHz: A trial test WiMAX based in rural environment. In: **Proceedings of the 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom 2007)**. Orland, FL, USA: [s.n.], 2007. p. 1–8. Cited on page 37.

ITU. **ITU-T P.800. Methods for Subjective Determination of Transmission Quality - Series P: Telephone Transmission Quality; Methods for Objective and Subjective Assessment of Quality**. 1996. Disponível em: <<http://goo.gl/dvboM>>. Cited on page 81.

IVORY, M. Y.; HEARST, M. A. The state of the art in automating usability evaluation of user interfaces. **ACM Comput. Surv.**, v. 33, n. 4, p. 470–516, 2001. Cited 2 times on pages 37 and 41.

JALOTE, P. **An Integrated Approach to Software Engineering**. third. New York, USA: Springer-Verlag New York, Inc., 2005. Cited on page 42.

JANSSEN, T.; ABREU, R.; GEMUND, A. van. Zoltar: A toolset for automatic fault localization. In: **Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)**. Auckland, New Zealand: [s.n.], 2009. p. 662–664. Cited on page 45.

JAVED, A. Z.; STROOPER, P.; WATSON, G. Automated generation of test cases using model-driven architecture. In: **Proceedings of the 2nd International Workshop on Automation of Software Test (AST 2007)**. Minneapolis, MN, USA: [s.n.], 2007. p. 1–3. Cited on page 46.

JEKOSCH, U. **Voice and speech quality perception**. Springer-Verlag Berlin and Heidelberg GmbH & Company KG, 2005. (Signals and communication technology). ISBN 9783540288602. Disponível em: <<http://books.google.com.br/books?id=8OM7CmWQYAcC>>. Cited 2 times on pages 74 and 80.

JIANG, M.; CHEN, T.; KUO, F.-C.; DING, Z. Testing central processing unit scheduling algorithms using metamorphic testing. In: **Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS 2013)**. Beijing, China: [s.n.], 2013. p. 530–536. Cited on page 47.

JIN, H.; WANG, Y.; CHEN, N.-W.; GOU, Z.; WANG, S. Artificial neural network for automatic test oracles generation. In: **Proceedings of the International Conference on Computer Science and Software Engineering (CSSE 2008)**. Wuhan, China: [s.n.], 2008. v. 2, p. 727–730. Cited 2 times on pages 60 and 61.

JUST, R.; SCHWEIGGERT, F. Automating software tests with partial oracles in integrated environments. In: **Proceedings of the 5th Workshop on Automation of Software Test (AST 2010)**. Cape Town, South Africa: [s.n.], 2010. p. 91–94. Cited on page 50.

K, M.; P, V. Hindi-english speech-to-speech translation system for travel expressions. In: **Computation of Power, Energy Information and Commuincation (ICCPEIC), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 0250–0255. Cited on page 31.

KANEWALA, U.; BIEMAN, J. Techniques for testing scientific programs without an oracle. In: **Proceedings of the 5th International Workshop on Software Engineering for Computational Science and Engineering (SE-CSE 2013)**. San Francisco, CA, USA: [s.n.], 2013. p. 48–57. Cited on page 60.

KARABETSOS, S.; TSIAKOULIS, P.; CHALAMANDARIS, A.; RAPTIS, S. Embedded unit selection text-to-speech synthesis for mobile devices. **IEEE Transactions on Consumer Electronics**, v. 55, n. 2, p. 613–621, May 2009. ISSN 0098-3063. Cited on page 31.

KARAMAT, T.; JAMIL, A. N. Reducing test cost and improving documentation in tdd (test driven development). In: **Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)**. [S.l.: s.n.], 2006. p. 73–76. Cited on page 38.

KIM-PARK, D.; RIVA, C. de la; TUYA, J. An automated test oracle for XML processing programs. In: **Proceedings of the 1st International Workshop on Software Test Output Validation (STOV 2010)**. Trento, Italy: [s.n.], 2010. p. 5–12. Cited on page 45.

KIRKPATRICK, B.; O'BRIEN, D.; SCAIFE, R. A comparison of spectral continuity measures as a join cost in concatenative speech synthesis. In: **Proceedings of the 18th IET Irish Signals and Systems Conference**. [S.l.: s.n.], 2006. p. 515–520. Cited on page 88.

KITCHENHAM, B.; CHARTERS, S.; BUDGEN, D.; BRERETON, P.; TURNER, M.; LINKMAN, S.; JORGENSEN, M.; MENDES, E.; VISAGGIO, G. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. [S.l.], 2007. Version 2.3. Cited on page 77.

KITCHENHAM, B.; PRETORIUS, R.; BUDGEN, D.; BRERETON, O. P.; TURNER, M.; NIAZI, M.; LINKMAN, S. Systematic literature reviews in software engineering – a tertiary study. **Information and Software Technology**, v. 52, n. 8, p. 792 – 805, 2010. Cited on page 63.

KNIGHT, J. C.; LEVESON, N. G. An experimental evaluation of the assumption of independence in multi-version programming. **IEEE Transactions on Software Engineering**, v. 12, p. 96–109, 1986. Cited on page 61.

KOWALCZYK, E.; MEMON, A. Extending manual gui testing beyond defects by building mental models of software behavior. In: **2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)**. [S.l.: s.n.], 2015. p. 35–41. Cited on page 68.

KREKOVIC, G.; PRENNER, V. A complete croatian language text-to-speech system. In: **Proceedings of the International Symposium (ELMAR)**. [S.l.: s.n.], 2010. p. 351–354. ISSN 1334-2630. Cited on page [81](#).

KUMAR, M.; SHARMA, A.; GARG, S. A study of aspect oriented testing techniques. In: **Proceedings of the IEEE Symposium on Industrial Electronics Applications (ISIEA 2009)**. Kuala Lumpur, Malaysia: [s.n.], 2009. p. 996–1001. Cited on page [36](#).

KUO, F.-C.; CHEN, T. Y.; TAM, W. Testing embedded software by metamorphic testing: A wireless metering system case study. In: **Proceedings of the 36th IEEE Conference on Local Computer Networks (LCN 2011)**. Bonn, Germany: [s.n.], 2011. p. 291–294. Cited on page [60](#).

KWON, B. C.; KIM, H.; WALL, E.; CHOO, J.; PARK, H.; ENDERT, A. Axisketcher: Interactive nonlinear axis mapping of visualizations through user drawings. **IEEE Transactions on Visualization and Computer Graphics**, v. 23, n. 1, p. 221–230, Jan 2017. ISSN 1077-2626. Cited on page [68](#).

LASALLE, J.; PEUREUX, F.; GUILLET, J. Automatic test concretization to supply end-to-end MBT for automotive mechatronic systems. In: **Proceedings of the 1st International Workshop on End-to-End Test Script Engineering (ETSE 2011)**. Toronto, Canada: [s.n.], 2011. p. 16–23. Cited on page [57](#).

LATOUI, A.; DJAHLI, F. An optical BILBO for online testing of embedded systems. **IEEE Design Test**, v. 30, n. 3, p. 34–48, 2013. Cited on page [36](#).

LEE, C. H.; JUNG, S. K.; KANG, H. G. Applying a speaker-dependent speech compression technique to concatenative tts synthesizers. **IEEE Transactions on Audio, Speech, and Language Processing**, v. 15, n. 2, p. 632–640, Feb 2007. ISSN 1558-7916. Cited on page [31](#).

LEOTTA, M.; CLERISSI, D.; RICCA, F.; TONELLA, P. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In: **Proceedings of the 20th Working Conference on Reverse Engineering (WCRE 2013)**. Koblenz, Germany: [s.n.], 2013. p. 272–281. Cited on page [36](#).

LIKE, P.; SAHAMA, T.; WINDSOR, C.; HUXLEY, C. Designing a quality health information system: Case study analysis in papua new guinea. In: **2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)**. [S.l.: s.n.], 2015. p. 270–270. Cited on page [68](#).

LIN, J.; HO, I. A new perspective on formal testing method for real-time software. In: **Proceedings of the 26th Euromicro Conference 2000 (Euromicro 2000)**. Maastricht, Netherlands: [s.n.], 2000. v. 2, p. 270–276. Cited on page [58](#).

_____. Generating timed test cases with oracles for real-time software. **Advances in Engineering Software**, v. 32, n. 9, p. 705–715, 2001. Cited on page [58](#).

LIN, T. W.; HUANG, P. Y.; CHENG, C. W. C. Computer-aided diagnosis in medical imaging: Review of legal barriers to entry for the commercial systems. In: **2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)**. [S.l.: s.n.], 2016. p. 1–5. Cited on page [68](#).

LIU, H.; KUO, F.-C.; TOWEY, D.; CHEN, T. Y. How effectively does metamorphic testing alleviate the oracle problem? **IEEE Transactions on Software Engineering**, v. 40, n. 1, p. 4–22, Jan 2014. Cited on page 60.

LUCKHAM, D.; HENKE, F. V. An overview of Anna, a specification language for Ada. **IEEE Software**, v. 2, n. 2, p. 9–22, 1985. Cited on page 54.

MA, J.; ZHANG, Y. Using topic model for intelligent computer-aided diagnosis in traditional chinese medicine. In: **2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)**. [S.l.: s.n.], 2016. p. 504–507. Cited on page 68.

MACHADO, P. D. L. **Testing from Structured Algebraic Specifications: The Oracle Problem**. Tese (Doutorado) — University of Edinburgh, 2000. Cited 3 times on pages 42, 44, and 47.

MANOLACHE, L.; KOURIE, D. Software testing using model programs. **Software: Practice and Experience**, v. 31, n. 13, p. 1211–1236, 2001. Cited on page 61.

MAO, Y.; BOQIN, F.; LI, Z.; YAO, L. Neural networks based automated test oracle for software testing. In: **Proceedings of the 13th International Conference on Neural Information Processing (ICONIP 2006)**. Berlin, Heidelberg: [s.n.], 2006. v. 3, p. 498–507. Cited on page 60.

MARKOVIC, D.; JOVANOVIĆ, Z.; PESOVIC, U.; RANĐIĆ, S. Test-driven development (tdd) in development of measure application and transducers services. In: **Telecommunications Forum (TELFOR), 2011 19th**. [S.l.: s.n.], 2011. p. 1316–1319. Cited on page 38.

MATHUR, A. P. **Foundations of Software Testing**. first. Boston, USA: Addison-Wesley Professional, 2007. Cited 3 times on pages 35, 36, and 37.

MCDONALD, J.; HOFFMAN, D.; STROOPER, P. Programmatic testing of the standard template library containers. In: **Proceedings of the 13th IEEE International Conference on Automated Software Engineering (ASE 1998)**. Honolulu, HI, USA: [s.n.], 1998. p. 147–156. Cited on page 50.

MCDONALD, J.; STROOPER, P. Translating Object-Z specifications to passive test oracles. In: **Proceedings of the 2nd IEEE International Conference on Formal Engineering Methods (ICEFEM 1998)**. Brisbane, Australia: [s.n.], 1998. p. 165–175. Cited on page 47.

MCDONALD, J.; STROOPER, P.; HOFFMAN, D. Tool support for generating passive C++ test oracles from object-z specifications. In: **Proceedings of the 10th Asia-Pacific Software Engineering Conference Software Engineering Conference (APSEC 2003)**. Chiang Mai, Thailand: [s.n.], 2003. p. 322–332. Cited on page 50.

MCENNIS D., C. M.; FUJINAGA., I. jaudio: Additions and improvements. In: **Proceedings of the International Conference on Music Information Retrieval**. [S.l.: s.n.], 2006. p. 385–6. Cited 2 times on pages 99 and 103.

MCENNIS D., C. M. I. F.; DEPALLE, P. jaudio: A feature extraction library. In: **Proceedings of the International Conference on Music Information Retrieval**. [S.l.: s.n.], 2005. p. 600–603. Cited 2 times on pages 99 and 103.

MCCLOUGHLIN, I. **Applied Speech and Audio Processing: With Matlab Examples**. 1st. ed. New York, NY, USA: Cambridge University Press, 2009. ISBN 0521519543, 9780521519540. Cited on page 74.

MCMASTER, S.; YUAN, X. Developing a feedback-driven automated testing tool for Web applications. In: **Proceedings of the 12th International Conference on Quality Software (QSIC 2012)**. Xi'an, China: [s.n.], 2012. p. 210–213. Cited on page 36.

MCMINN, P.; SHAHBAZ, M.; STEVENSON, M. Search-based test input generation for string data types using the results of Web queries. In: **Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)**. Montreal, Canada: [s.n.], 2012. p. 141–150. Cited on page 45.

MCMINN, P.; STEVENSON, M.; HARMAN, M. Reducing qualitative human oracle costs associated with automatically generated test data. In: **Proceedings of the 1st International Workshop on Software Test Output Validation (STOV 2010)**. Trento, Italy: [s.n.], 2010. p. 1–4. Cited 2 times on pages 43 and 45.

MELLO, C. A. **Processamento Digital de Sinais – Notas de aula**. 2013. Disponível em: <<http://www.cin.ufpe.br/~cabm/pds/PDS.pdf>>. Cited 2 times on pages 75 and 76.

MEMON, A.; BANERJEE, I.; NAGARAJAN, A. What test oracle should I use for effective GUI testing? In: **Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE 2003)**. Montreal, Canada: [s.n.], 2003. p. 164–173. Cited 2 times on pages 53 and 61.

MEMON, A.; XIE, Q. Empirical evaluation of the fault-detection effectiveness of smoke regression test cases for GUI-based software. In: **Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM 2004)**. Chicago, IL, USA: [s.n.], 2004. p. 8–17. Cited on page 47.

MILLER, T.; STROOPER, P. Supporting the software testing process through specification animation. In: **Proceedings of the 1st International Conference on Software Engineering and Formal Methods (SEFM 2003)**. Brisbane, Australia: [s.n.], 2003. p. 14–23. Cited on page 50.

MIRANDA, I. I.; MEIRELES, A. Análise acústico-comparativa de vogais brasileiras com vogais norte-americanas. In: **Proceedings of the I Congresso Nacional de Estudos Linguísticos**. Vitória/ES: [s.n.], 2011. p. 1–10. Cited 2 times on pages 109 and 110.

MUCCINI, H.; FRANCESCO, A. D.; ESPOSITO, P. Software testing of mobile applications: Challenges and future research directions. In: **Automation of Software Test (AST), 2012 7th International Workshop on**. [S.l.: s.n.], 2012. p. 29–35. Cited on page 27.

MURPHY, C.; KAISER, G. **Metamorphic Runtime Checking of Non-Testable Programs**. [S.l.], 2009. <<http://bit.ly/1p8rGfR>> - access: 12/09/2013. Cited on page 54.

MURPHY, C.; KAISER, G. E.; HU, L.; WU, L. Properties of machine learning applications for use in metamorphic testing. In: **Proceedings of the 20th International Conference on Software Engineering & Knowledge Engineering (SEKE 2008)**. San Francisco, CA, USA: [s.n.], 2008. p. 867–872. Cited on page 60.

MURPHY, C.; RAUNAK, M.; KING, A.; CHEN, S.; IMBRIANO, C.; KAISER, G.; LEE, I.; SOKOLSKY, O.; CLARKE, L.; OSTERWEIL, L. On effective testing of health care simulation software. In: **Proceedings of the 3rd Workshop on Software Engineering in Health Care (SEHC 2011)**. Waikiki, HI, USA: [s.n.], 2011. p. 40–47. Cited on page 60.

MURPHY, C.; SHEN, K.; KAISER, G. Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles. In: **Proceedings of the International Conference on Software Testing Verification and Validation (ICST 2009)**. Denver, CO, USA: [s.n.], 2009. p. 436–445. Cited on page 47.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. 3rd. ed. [S.l.]: Wiley Publishing, 2011. ISBN 1118031962. Cited 5 times on pages 29, 35, 36, 37, and 39.

NAGOWAH, L.; DOORGAH, K. Improving test data management in record and playback testing tools. In: **Computer Information Science (ICCIS), 2012 International Conference on**. [S.l.: s.n.], 2012. v. 2, p. 931–937. Cited on page 38.

NARDI, P. **On test oracles for Simulink-like models**. Tese (Doutorado) — University São Paulo, 2013. Cited 3 times on pages 47, 57, and 59.

NARDI, P.; DELAMARO, M. **Test oracles associated with dynamical system models**. [S.l.], 2011. <<http://bit.ly/1110mCW>>- Access: 12/18/2013. Cited on page 53.

NARDO, D. D.; PASTORE, F.; BRIAND, L. Generating complex and faulty test data through model-based mutation analysis. In: **2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)**. [S.l.: s.n.], 2015. p. 1–10. ISSN 2159-4848. Cited on page 67.

NORRENBROCK, C.; HINTERLEITNER, F.; HEUTE, U.; MOLLER, S. Instrumental assessment of prosodic quality for text-to-speech signals. **Signal Processing Letters, IEEE**, v. 19, n. 5, p. 255–258, 2012. ISSN 1070-9908. Cited on page 81.

OLIVEIRA, R.; DELAMARO, M.; NUNES, F. Exploring the O-Flm framework to support the test of programs with GUIs. In: **Proceedings of the 4th Brazilian Workshop on Systematic and Automated Software Testing (SAST 2010)**. Natal, Brasil: [s.n.], 2010. p. 31–40. Cited 3 times on pages 43, 45, and 47.

OLIVEIRA, R. A. Automating test oracles for systems with complex outputs. In: **Proceedings of the 7th Workshops on IEEE International Conference on Software Testing, Verification and Validation (WICST)**. Cleveland, OH, USA: [s.n.], 2014. p. 239–240. Cited on page 29.

OLIVEIRA, R. A. P.; DELAMARO, M. E.; NUNES, F. L. S. Exploring the O-Flm framework to Support the Test of Programs with GUIs. In: **Proceedings of the 4th Brazilian Workshop on Systematic and Automated Software Testing (SAST 2010)**. Natal, RN, Brasil: [s.n.], 2010. p. 31–40. Cited on page 30.

_____. **Testing techniques to TTS applications: a Systematic Mapping**. [S.l.], 2013. Cited on page 78.

OLIVEIRA, R. A. P.; KANEWALA, U.; NARDI, P. A. Chapter three - automated test oracles: State of the art, taxonomies, and trends. In: MEMON, A. (Ed.). [S.l.]: Elsevier, 2014, (Advances in Computers, v. 95). p. 113 – 199. Cited 4 times on pages 23, 53, 62, and 65.

OSGOOD, C. E. The nature and measurement of meaning. **Psychological Bulletin**, v. 49, n. 3, p. 197–237, 1952. Cited 2 times on pages 80 and 81.

PACKEVICUS, S.; ANDREJ, U.; BAREISA, E. Software testing using imprecise OCL constraints as oracles. In: **Proceedings of the 2007 International Conference on Computer systems and technologies (CompSysTech 2007)**. Ruse, Bulgaria: [s.n.], 2007. p. 1–6. Cited on page 56.

PAQUERAULT, S.; YARUSSO, L. M.; PAPAIOANNOU, J.; JIANG, Y.; NISHIKAWA, R. M. Radial gradient-based segmentation of mammographic microcalcifications: Observer evaluation and effect on CAD performance. **Medical Physics**, Medical Physics, v. 31, n. 9, p. 2648–2657, september 2004. Cited on page 91.

PASALA, A.; RAO, S.; GUPTA, A.; GUNTURU, S. On the validation of API execution-sequence to assess the correctness of application upon COTS upgrades deployment. In: **Proceedings of the 6th International IEEE Conference on Commercial-off-the-Shelf(COTS)-Based Software Systems (ICCBSS 2007)**. Banff, Canada: [s.n.], 2007. p. 225–232. Cited on page 50.

PHAN, L. P.; HUYNH, H. X.; HUYNH, H. H.; NGUYEN, K. M. Association-based recommender system using statistical implicative cohesion measure. In: **2016 Eighth International Conference on Knowledge and Systems Engineering (KSE)**. [S.l.: s.n.], 2016. p. 144–149. Cited on page 68.

PORTELE, T.; KRAMER, J. Adapting a tts system to a reading machine for the blind. In: **Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on**. [S.l.: s.n.], 1996. v. 1, p. 184–187 vol.1. Cited on page 31.

PRAPHAMONTRIPONG, U. Web mutation testing. In: **Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)**. Berlin, Germany: [s.n.], 2012. p. 495–498. Cited on page 36.

PRESSMAN, R. **Software Engineering: A Practitioner's Approach**. McGraw-Hill Education, 2010. (McGraw-Hill higher education). ISBN 9780073375977. Disponível em: <<https://books.google.com.br/books?id=y4k\AQAAIAAJ>>. Cited on page 40.

QIAN, Y.; FAN, Y.; HU, W.; SOONG, F. K. On the training aspects of deep neural network (dnn) for parametric tts synthesis. In: **2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. [S.l.: s.n.], 2014. p. 3829–3833. ISSN 1520-6149. Cited on page 31.

RABINER, L.; SCHAFER, R. **Digital Processing of Speech Signals**. [S.l.]: Englewood Cliffs: Prentice Hall, 1978. Cited 2 times on pages 74 and 75.

RAMLER, R.; MOSER, M.; PICHLER, J. Automated static analysis of unit test code. In: **2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)**. [S.l.: s.n.], 2016. v. 2, p. 25–28. Cited on page 38.

RAN, L.; DYRESON, C.; ANDREWS, A.; BRYCE, R.; MALLERY, C. Building test cases and oracles to automate the testing of Web database applications. **Information and Software Technology**, v. 51, n. 2, p. 460–477, 2009. Cited on page 50.

RAUF, A.; ANWAR, S.; RAMZAN, M.; REHMAN, S.; SHAHID, A. Ontology driven semantic annotation based GUI testing. In: **Proceedings of the 6th International Conference on Emerging Technologies (ICET 2010)**. Islamabad, Pakistan: [s.n.], 2010. p. 261–264. Cited on page 45.

REPASI, T. Software testing - state of the art and current research challenges. In: **Proceedings of the 5th International Symposium on Applied Computational Intelligence and Informatics (SACI 2009)**. Timisoara, Romênia: [s.n.], 2009. p. 47–50. Cited on page 29.

ROSE, L.; POULDING, S. Efficient probabilistic testing of model transformations using search. In: **Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE 2013)**. San Francisco, CA, USA: [s.n.], 2013. p. 16–21. Cited on page 47.

RUSSO, I.; BEHLAU, M. **Percepção da fala: análise acústica do português brasileiro**. first. São Paulo/SP: Lovise, 1993. v. 1. Cited on page 108.

SAK, H.; GUNGOR, T.; SAFKAN, Y. A corpus-based concatenative speech synthesis system for turkish. **Turkish Journal of Electrical Engineering and Computer Sciences**, v. 14, n. 2, p. 209–223, 2006. Cited 2 times on pages 86 and 88.

SALEOUS, H.; SHAIKH, A.; GUPTA, R.; SAGAHYROON, A. Read2me: A cloud-based reading aid for the visually impaired. In: **2016 International Conference on Industrial Informatics and Computer Systems (CIICS)**. [S.l.: s.n.], 2016. p. 1–6. Cited on page 31.

SALIVAHANAN, S.; ALAN, O. **Digital signal processing**. [S.l.]: McGraw-Hill Education (India) Pvt Limited, 2000. University of California. ISBN 9780074639962. Cited on page 75.

SALMI, K.; MAGREZ, H.; ZIYYAT, A. Interactive simulation as a virtual tool in electromagnetics for online education. In: **2014 Third IEEE International Colloquium in Information Science and Technology (CIST)**. [S.l.: s.n.], 2014. p. 192–195. ISSN 2327-185X. Cited on page 68.

SAMMOUDA, R.; ALRJIOUB, A. Mobile blind navigation system using rfid. In: **Computer Information Technology (GSCIT), 2015 Global Summit on**. [S.l.: s.n.], 2015. p. 1–4. Cited on page 31.

SANGWAN, O.; BHATIA, P.; SINGH, Y. Radial basis function neural network based approach to test oracle. **ACM SIGSOFT Software Engineering Notes**, v. 36, n. 5, p. 1–5, 2011. Cited on page 45.

SANKAR, S.; GOYAL, A.; SIKCHI, P. **Software Testing Using Algebraic Specification Based Test Oracles**. [S.l.], 1993. <<http://1.usa.gov/Oz2Avd>>- Acesses: 12/15/2013. Cited on page 54.

SBC. **II Seminário sobre Grandes Desafios da Computação no Brasil – Integração com a Indústria e uma Perspectiva para 2020**. Manaus, AM, Brasil: [s.n.], 2009. Disponível em: <<http://goo.gl/QMkYv>>. Cited on page 29.

SCHMANDT, C. **Voice communication with computers: conversational systems**. [S.l.]: Van Nostrand Reinhold, 1994. (VNR computer library). ISBN 9780442239350. Cited on page 74.

SEO, K. I.; CHOI, E. M. Rigorous vertical software system testing in IDE. In: **Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)**. Busan, Korea: [s.n.], 2007. p. 847–854. Cited on page 37.

SHAHAMIRI, S.; KADIR, W.; HASHIM, S. A comparative study on automated software test oracle methods. In: **Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA 2009)**. Porto, Portugal: [s.n.], 2009. p. 140–145. Cited 3 times on pages [41](#), [46](#), and [48](#).

SHAHAMIRI, S.; KADIR, W.; IBRAHIM, S. An automated oracle approach to test decision-making structures. In: **Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010)**. Chengdu, China: [s.n.], 2010. v. 5, p. 30–34. Cited on page [45](#).

_____. A single-network ANN-based oracle to verify logical software modules. In: **Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE 2010)**. San Juan, Puerto Rico: [s.n.], 2010. p. 272–276. Cited on page [45](#).

SHAMSHIRI, S.; JUST, R.; ROJAS, J. M.; FRASER, G.; MCMINN, P.; ARCURI, A. Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t). In: **Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)**. [S.l.: s.n.], 2015. p. 201–211. Cited on page [38](#).

SHU, G.; BAI, Z.; PODGURSKI, A. Statistical evaluation of complex input-output transformations. In: **2011 IEEE 22nd International Symposium on Software Reliability Engineering**. [S.l.: s.n.], 2011. p. 70–79. ISSN 1071-9458. Cited 2 times on pages [66](#) and [68](#).

SHUKLA, R.; STROOPER, P.; CARRINGTON, D. Tool support for statistical testing of software components. In: **Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC 2005)**. Taipei, Taiwan: [s.n.], 2005. p. 16–24. Cited 2 times on pages [50](#) and [55](#).

SMITH, J.; DULAY, N.; TOTH, M. A.; AMFT, O.; ZHANG, Y. Exploring concept drift using interactive simulations. In: **2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)**. [S.l.: s.n.], 2013. p. 49–54. Cited on page [68](#).

SNEED, H. State coverage of embedded realtime programs. In: **Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis (STVA 1988)**. Banff, Canada: [s.n.], 1988. p. 245–250. Cited on page [45](#).

SOMMERVILLE, I. **Software Engineering**. 9th. ed. USA: Addison-Wesley Publishing Company, 2010. ISBN 0137035152, 9780137035151. Cited on page [28](#).

SPAIS, X. P. I. A complemented greek text to speech system. **WSEAS Transactions on Systems**, v. 4, n. 1, p. 116–121, 2005. Cited on page [81](#).

SRINIVASAN, J.; LEVESON, N. Automated testing from specifications. In: **Proceedings of the 21st Digital Avionics Systems Conference (DASC 2002)**. Irvine, CA, USA: [s.n.], 2002. v. 1, p. 1–8. Cited on page [48](#).

STAATS, M.; GAY, G.; HEIMDAHL, M. Automated oracle creation support, or: How I learned to stop worrying about fault propagation and love mutation testing. In: **Proceedings of the 34th International Conference on Software Engineering (ICSE 2012)**. Zurich, Switzerland: [s.n.], 2012. p. 870–880. Cited 2 times on pages [45](#) and [48](#).

STAATS, M.; WHALEN, M.; HEIMDAHL, M. Programs, tests, and oracles: the foundations of testing revisited. In: **Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)**. Waikiki, HI, USA: [s.n.], 2011. p. 391–400. Cited 3 times on pages 44, 46, and 48.

SUN, J.; LI, G.; XU, C. Prior distribution of missile range trial successive test data based on random weighting. In: **Proceedings of the 4th International Conference on Computational and Information Sciences (ICCIS 2012)**. Kuala Lumpur, Malaysia: [s.n.], 2012. p. 1163–1166. Cited on page 37.

SURO, F.; ONO, Y. Japanese efl learners uses of text-to-speech technology and their learning behaviors: A pilot study. In: **2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)**. [S.l.: s.n.], 2016. p. 296–301. Cited on page 68.

SäGESSER, K.; JOSEPH, B.; GRAU, R. Introducing an iterative life-cycle model at credit suisse it switzerland. **IEEE Software**, v. 30, n. 2, p. 68–73, 2013. Cited on page 27.

TAFNER, M. A. **Reconhecimento de palavras isoladas usando redes neurais artificiais**. Dissertação (Mestrado em Engenharia de Produção) — Universidade Federal de Santa Catarina, Florianópolis/SC, 1996. Cited on page 108.

TAKAHASHI, J. An Automated Oracle for Verifying GUI Objects. **ACM SIGSOFT Software Engineering Notes**, ACM, New York, NY, USA, v. 26, n. 4, p. 83–88, july 2001. ISSN 0163-5948. Cited on page 42.

TAKAMICHI, S.; TODA, T.; BLACK, A. W.; NEUBIG, G.; SAKTI, S.; NAKAMURA, S. Post-filters to modify the modulation spectrum for statistical parametric speech synthesis. **IEEE/ACM Transactions on Audio, Speech, and Language Processing**, v. 24, n. 4, p. 755–767, April 2016. ISSN 2329-9290. Cited on page 31.

TAYLOR, P. **Text-to-Speech Synthesis**. 1. ed. [S.l.]: Cambridge University Press, 2009. ISBN 9780521899277. Cited 6 times on pages 31, 71, 72, 73, 76, and 77.

TEAM, J. **JUnit 4**. 2013. <<http://junit.org/>>. Accessed: 2013-12-15. Cited on page 46.

TORSEL, A.-M. Automated test case generation for Web applications from a domain specific model. In: **Proceedings of the 35th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW 2011)**. Munich, Germany: [s.n.], 2011. p. 137–142. Cited on page 48.

TURK, O.; SCHRODER, M. A comparison of voice conversion methods for transforming voice quality in emotional speech synthesis. In: **Proceedings of the 9th Annual Conference of the International Speech Communication Association (INTERSPEECH)**. [S.l.: s.n.], 2008. p. 2282–2285. Cited on page 81.

TURKMEN, H. I.; KARSLIGIL, M. E. Reconstruction of dysphonic speech for synthesizing normally phonated speech. In: **Proceedings of the 17th IEEE Signal Processing and Communications Applications Conference (SIU 2009)**. [S.l.: s.n.], 2009. p. 632–635. Cited on page 88.

UNGUREAN, C.; BURILEANU, D. An advanced nlp framework for high-quality text-to-speech synthesis. In: **Speech Technology and Human-Computer Dialogue (SpeD), 2011 6th Conference on**. [S.l.: s.n.], 2011. p. 1–6. Cited on page 71.

VADERNA, R.; DEJANOVIC, I.; MILOSAVLJEVIC, G. Grad: A new graph drawing and analysis library. In: **2016 Federated Conference on Computer Science and Information Systems (FedCSIS)**. [S.l.: s.n.], 2016. p. 1597–1602. Cited on page 68.

VANMALI, M.; LAST, M.; KANDEL, A. Using a neural network in the software testing process. **International Journal of Intelligent Systems**, v. 17, n. 1, p. 45–62, 2002. Cited on page 45.

VAZQUEZ, D.; XU, J.; RAMOS, S.; LOPEZ, A.; PONSÁ, D. Weakly supervised automatic annotation of pedestrian bounding boxes. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2013)**. Portland, OR, USA: [s.n.], 2013. p. 706–711. Cited on page 48.

VIARO, M. E.; FILHO, Z. O. G. Análise quantitativa da frequência dos fonemas e estruturas silábicas portuguesas. In: **Estudos Linguísticos XXXVI**. [S.l.: s.n.], 2007. v. 36, p. 27 – 36. Cited on page 110.

VISWANATHAN, M.; VISWANATHAN, M. Measuring speech quality for text-to-speech systems: development and assessment of a modified mean opinion score (mos) scale. **Computer Speech & Language**, v. 19, n. 1, p. 55–83, 2005. ISSN 0885-2308. Cited 3 times on pages 80, 81, and 82.

WAN, V.; LATORRE, J.; YANAGISAWA, K.; BRAUNSCHWEILER, N.; CHEN, L.; GALES, M. J. F.; AKAMINE, M. Building hmm-tts voices on diverse data. **IEEE Journal of Selected Topics in Signal Processing**, v. 8, n. 2, p. 296–306, April 2014. ISSN 1932-4553. Cited on page 31.

WANG, F.; WU, J.-H.; HUANG, C.-H.; CHANG, K.-H. Evolving a test oracle in black-box testing. In: **Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE 2011)**. Saarbrücken, Germany: [s.n.], 2011. p. 310–325. Cited on page 60.

WANG, F.; YAO, L.-W.; WU, J.-H. Intelligent test oracle construction for reactive systems without explicit specifications. In: **Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2011)**. Sydney, Australia: [s.n.], 2011. p. 89–96. Cited on page 60.

WANG, X.; QI, Z.-C.; LI, S. An optimized method for automatic test oracle generation from real-time specification. In: **Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005)**. Shanghai, China: [s.n.], 2005. p. 440–449. Cited on page 58.

WEYUKER, E. On testing non-testable programs. **The Computer Journal**, v. 25, n. 4, p. 465–470, 1982. Cited 6 times on pages 41, 44, 46, 47, 48, and 49.

XIE, X.; HO, J. W. K.; MURPHY, C.; KAISER, G.; XU, B.; CHEN, T. Y. Testing and validating machine learning classifiers by metamorphic testing. **Jornal of System and Software**, v. 84, n. 4, p. 544–558, 2011. Cited on page 60.

XIE, X.; WONG, W. E.; CHEN, T. Y.; XU, B. Metamorphic slice: An application in spectrum-based fault localization. **Information and Software Technology**, v. 55, n. 5, p. 866–879, 2013. Cited on page 60.

YIN, Y.; LIU, B.; NI, H. Real-time embedded software testing method based on extended finite state machine. **Systems Engineering and Electronics, Journal of**, v. 23, n. 2, p. 276–285, 2012. Cited on page [36](#).

ZARAKET, F.; MASRI, W.; ADAM, M.; HAMMOUD, D.; HAMZEH, R.; FARHAT, R.; KHAMISSI, E.; NOUJAIM, J. GUICOP: Specification-based GUI testing. In: **Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST 2012)**. Montreal, Canada: [s.n.], 2012. p. 747–751. Cited on page [60](#).

ZHANG, Z.; CHAN, W.; TSE, T.; HU, P. Experimental study to compare the use of metamorphic testing and assertion checking. **Journal of Software**, v. 20, n. 10, p. 2637–2654, 2009. Cited on page [47](#).

ZHOU, S.-S.; CHEN, Q.-C.; WANG, D.-D.; YANG, X.-H. A corpus-based concatenative mandarin singing voice synthesis system. In: **Proceedings of the 7th International Conference on Machine Learning and Cybernetics (ICMLC 2008)**. [S.l.: s.n.], 2008. v. 5, p. 2695–2699. Cited on page [80](#).

RESULTS OF FEATURE-BASED TEST ORACLES EVALUATING NORMALIZATION ISSUES IN TTS SYSTEMS (SEC. 5.3)

Find bellow all of the tables regarding some oracle configuration and a given TTS.

Table 32 – Oracle 01 – CPqD – Feature-based test oracles against normalization issues

Sent	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7	Ex 8	Ex 9	Ex 10	Ex 11	Ex 12	Ex 13	Ex 14	Ex 15	Ex 16	Ex 17	Enc. Dist.
Phr01-Norm	0.9609	0.9305	0.6637	0.9857	0.7259	0.7391	0.9764	0.9752	0.6989	0.5488	0.9125	0.9753	0.9609	0.9480	0.9764	0.9168	0.9528	0.0346
Phr01-Real	0.9584	0.9269	0.6679	0.9862	0.7254	0.7392	0.9771	0.9681	0.6994	0.5491	0.9125	0.9682	0.9584	0.9677	0.9899	0.9386	0.9536	
Phr02-Norm	0.7845	0.8043	0.7598	0.9689	0.7721	0.7920	0.9796	0.8077	0.6882	0.5582	0.9152	0.8083	0.7843	0.7528	0.6915	0.7104	0.9538	0.1456
Phr02-Real	0.8296	0.8353	0.7503	0.9792	0.7590	0.7751	1.0000	0.8383	0.7590	0.6234	0.9252	0.6396	0.8947	0.7961	0.7492	0.7825	0.9693	
Phr03-Norm	0.5921	0.6160	0.8098	0.9802	0.8661	0.8721	0.9565	0.8721	0.6346	0.5247	0.9495	0.8385	0.8295	0.7961	0.7492	0.7825	0.9693	
Phr03-Real	0.6395	0.6444	0.8650	0.9783	0.9067	0.9051	0.9147	0.6550	0.7999	0.6746	0.9216	0.6552	0.6394	0.6380	0.5058	0.5441	0.8715	0.1558
Phr04-Norm	0.7751	0.8286	0.7434	0.9842	0.8941	0.8928	0.9345	0.8225	0.9612	0.7187	0.8921	0.8226	0.7749	0.6569	0.7050	0.8139	0.9311	0.1154
Phr04-Real	0.7298	0.7963	0.7625	0.9758	0.9182	0.9182	0.9137	0.8043	0.7963	0.7673	0.8948	0.8019	0.7295	0.6338	0.7539	0.8227	0.9444	
Phr05-Norm	0.6742	0.6718	0.7003	0.9748	0.8142	0.8358	0.9558	0.7013	0.8932	0.7084	0.8785	0.7046	0.6740	0.6601	0.5907	0.6772	0.9244	0.1284
Phr05-Real	0.6830	0.6798	0.7774	0.9664	0.8430	0.8638	0.9343	0.7081	0.8599	0.7581	0.8723	0.7084	0.6828	0.6848	0.5643	0.6401	0.8814	
Phr06-Norm	0.4248	0.4690	1.0000	1.0000	1.0000	1.0000	0.9130	0.4584	0.9918	1.0000	0.8918	0.4586	0.4247	0.3911	0.2367	0.2779	0.8667	0.5120
Phr06-Real	0.4554	0.4694	0.8050	0.9931	0.8297	0.8165	0.9353	0.4936	0.8814	0.6608	0.9657	0.4928	0.4552	0.4500	0.3434	0.3583	0.9351	
Phr07-Norm	0.8648	0.8995	0.7827	0.9827	0.8078	0.8259	0.9130	0.8942	0.8079	0.6464	0.8521	0.8942	0.8644	0.7906	0.8171	0.8919	0.8885	0.4455
Phr07-Real	1.0000	1.0000	0.7200	0.9728	0.7503	0.7703	0.9390	0.8745	0.5629	0.5629	0.9793	1.0000	1.0000	1.0000	1.0000	1.0000	0.9290	
Phr08-Norm	0.7735	0.7852	0.7513	0.9936	0.8318	0.8468	0.9936	0.8001	0.8002	0.6884	0.9701	0.8003	0.7732	0.7282	0.6903	0.7221	0.9221	0.1240
Phr08-Real	0.7579	0.7702	0.8055	0.9886	0.8739	0.8821	0.9308	0.8021	0.8171	0.6861	0.9188	0.8027	0.7578	0.6894	0.6581	0.6534	0.9024	
Phr09-Norm	0.6898	0.6820	0.8565	0.9812	0.9435	0.9412	0.8612	0.6978	1.0000	0.8545	0.9152	0.6981	0.6897	0.6590	0.5779	0.5568	0.8885	0.3679
Phr09-Real	0.7687	0.7516	0.9277	0.9822	0.8588	0.8792	0.9607	0.7768	0.8996	0.7128	0.8875	0.7085	0.6562	0.5548	0.5647	0.7172	0.9223	
Phr10-Norm	0.7543	0.8264	0.7412	0.9837	0.8542	0.8603	0.9535	0.7916	0.7660	0.6393	0.8853	0.7919	0.7541	0.6662	0.7403	0.8280	0.9074	0.1782
Phr10-Real	0.7968	0.8295	0.7173	0.9807	0.8098	0.8144	0.9771	0.8201	0.8143	0.5714	0.9432	0.8202	0.7965	0.7544	0.7783	0.8239	0.9496	
Phr11-Norm	0.6570	0.7139	0.7609	0.9773	0.7799	0.7836	0.9472	0.7109	0.7482	0.6114	0.8870	0.7112	0.6568	0.5582	0.5783	0.7464	0.9198	0.0359
Phr11-Real	0.6563	0.7117	0.7641	0.9896	0.7790	0.7844	0.9462	0.7055	0.7468	0.6129	0.8875	0.7058	0.6562	0.5548	0.5647	0.7172	0.9223	
Phr12-Norm	0.6574	0.6678	0.7237	0.9783	0.7799	0.7825	0.9256	0.6897	0.8079	0.5617	0.9279	0.6900	0.6572	0.6809	0.6988	0.6598	0.9496	
Phr12-Real	0.6586	0.6696	0.7248	0.9728	0.7805	0.7812	0.9269	0.6898	0.8107	0.5603	0.9279	0.6900	0.6584	0.6962	0.7004	0.6511	0.9469	0.0193
Phr13-Norm	0.8005	0.8118	0.6679	0.9758	0.7946	0.7992	0.9519	0.8135	0.8148	0.6038	0.9062	0.8138	0.8003	0.7496	0.6946	0.7908	0.9633	0.1898
Phr13-Real	0.8288	0.8485	0.7806	0.9728	0.8576	0.8545	0.9455	0.8524	0.7974	0.6449	0.9010	0.8528	0.8286	0.7394	0.7291	0.8660	0.9271	
Phr14-Norm	0.6276	0.6811	0.7646	0.9792	0.8914	0.8986	0.9179	0.6583	0.9193	0.7725	0.8823	0.6585	0.6275	0.5661	0.5605	0.7411	0.9066	0.0279
Phr14-Real	0.6280	0.6816	0.7635	0.9812	0.8921	0.8985	0.9172	0.6697	0.9215	0.7722	0.8825	0.6700	0.6279	0.5842	0.5531	0.7519	0.9080	
Phr15-Norm	0.4718	0.5292	0.6769	0.9911	0.8218	0.8228	0.9000	0.5001	0.7979	0.5811	0.8888	0.5002	0.4716	0.4509	0.3693	0.4558	0.9196	0.1528
Phr15-Real	0.4414	0.5000	0.6291	0.9886	0.7626	0.7630	0.9231	0.4731	0.8198	0.5594	0.9243	0.4733	0.4414	0.4217	0.3317	0.4325	0.9849	
Phr16-Norm	0.6359	0.6612	0.7168	0.9812	0.8573	0.8664	0.9102	0.6540	0.9088	0.6752	0.9252	0.6543	0.6359	0.5934	0.5333	0.5832	0.9337	0.2209
Phr16-Real	0.5702	0.6156	0.7088	0.9713	0.8717	0.8786	0.8974	0.5888	0.9129	0.6943	0.9297	0.5890	0.5701	0.5220	0.4446	0.4589	0.9248	
Phr17-Norm	0.6318	0.6709	0.7901	0.9708	0.7829	0.7881	0.9130	0.6689	0.9685	0.5517	0.9351	0.6692	0.6316	0.5910	0.5465	0.6726	0.9206	0.1216
Phr17-Real	0.6097	0.6577	0.8480	0.9792	0.8324	0.8337	0.9196	0.6568	0.9804	0.5958	0.9243	0.6570	0.6097	0.5824	0.5159	0.6303	0.9020	
Phr18-Norm	0.8161	0.8654	0.6823	0.9827	0.7610	0.7626	0.9640	0.8505	0.7286	0.5215	0.8841	0.8504	0.8158	0.7710	0.8628	0.9882	0.9389	0.1543
Phr18-Real	0.7809	0.8317	0.7046	0.9758	0.7801	0.7817	0.9521	0.8109	0.7518	0.5300	0.9044	0.8115	0.7807	0.7373	0.7907	0.8968	0.9425	
Phr19-Norm	0.4706	0.5372	0.7242	0.9758	0.8757	0.8851	0.9231	0.5085	0.8837	0.7049	0.9270	0.5088	0.4705	0.4475	0.3649	0.4215	0.9565	0.0178
Phr19-Real	0.4760	0.5394	0.7306	0.9787	0.8754	0.8850	0.9240	0.5076	0.8818	0.7046	0.9261	0.5078	0.4759	0.4598	0.3705	0.4198	0.9601	
Phr20-Norm	0.6780	0.7547	0.8119	0.9862	0.8780	0.8904	0.8803	0.7322	0.9311	0.7319	0.9216	0.7325	0.6780	0.5580	0.6733	0.6857	0.9106	0.1574
Phr20-Real	0.6815	0.7387	0.8119	0.9822	0.8510	0.8528	0.9112	0.7378	0.8978	0.6108	0.9197	0.7380	0.6814	0.6228	0.7070	0.6718	0.9139	
Phr21-Norm	0.6751	0.7055	0.8406	0.9876	0.8371	0.8584	0.9575	0.7074	0.9147	0.6802	0.9702	0.7077	0.6749	0.6150	0.5593	0.7772	0.9349	0.1742
Phr21-Real	0.7225	0.7547	0.7545	0.9852	0.8181	0.8463	0.9659	0.7429	0.9325	0.6511	1.0000	0.7432	0.7224	0.6456	0.6457	0.8250	0.9479	
Phr22-Norm	0.5388	0.5784	0.7625	0.9837	0.8288	0.8299	0.9432	0.5617	0.7847	0.5929	0.9107	0.5619	0.5386	0.4926	0.4271	0.4890	0.9062	0.0476
Phr22-Real	0.5398	0.5859	0.7418	0.9832	0.8190	0.8197	0.9432	0.5761	0.7664	0.5782	0.9125	0.5764	0.5399	0.4388	0.4704	0.4893	0.9063	
Phr23-Norm	0.5688	0.5908	0.6658	0.9689	0.7472	0.7457	0.9706	0.8714	0.5500	0.9793	0.5878	0.5886	0.5306	0.5105	0.6766	1.0000	0.1787	
Phr23-Real	0.5960	0.6156	0.7242	0.9881	0.7990	0.8133	0.9727	0.6238	0.9197	0.5761	0.9729	0.6240	0.5959	0.5546	0.5535	0.6958	0.8984	
Phr24-Norm	0.6237	0.6732	0.7699	0.9827	0.7854	0.7900	0.9813	0.6866	0.7897	0.5279	0.8815	0.6869	0.6236	0.5302	0.5372	0.6389	0.9198	0.0916
Phr24-Real	0.6072	0.6550	0.7439	0.9802	0.7556	0.7599	0.9703	0.8002	0.8002	0.5182	0.9005	0.6543	0.6072	0.5322	0.5089	0.6215	0.9490	
Phr25-Norm	0.8958	0.9035	0.8735	0.9674	0.8267	0.8441	0.9360	0.8987	0.9471	0.6417	0.8862	0.8990	0.8960	0.8654	0.8624	0.8925	0.8910	0.1620
Phr25-Real	0.9395	0.9105	0.8948	0.9871	0.8384	0.8450	0.9243	0.9405	0.9466	0.6382	0.8759	0.9411	0.9393	0.9047	0.7783	0.7983	0.8782	0.0781
Phr26-Norm	0.9902	0.9681	0.6371	0.9644	0.7763	0.7947	0.9186	0.9921	0.9535	0.5720	0.9225	0.9928	0.9900	0.9693	0.8407	0.8768	0.9376	0.0329
Phr26-Real	0.9927	0.9708	0.6366	0.9723	0.7768	0.7956	0.9186	0.9970	0.9544	0.5732	0.9216	0.9968	0.9925	0.9717	0.8647	0.8961	0.9391	
Phr27-Norm	0.6362	0.7037	0.8342	0.9812	0.8420	0.8467	0.9430	0.6924	0.9407	0.6223	0.9179	0.6927	0.6361	0.5334	0.5574	0.7196	0.9215	
Phr27-Real	0.6424	0.6953	0.8045	0.9871	0.8224	0.8244	0.9479	0.6928	0.9494	0.6999	0.9197	0.6931	0.6422	0.5769	0.5671	0.6761	0.9250	0.0781
Phr28-Norm	0.5941	0.6001	0.6998	0.9654	0.8088	0.8078	0.9493	0.6185	0.9503	0.6170	0.9225	0.6187	0.5939	0.5494	0.4415	0.5044	0.9288	0.2955
Phr28-Real	0.6189	0.6147	0.9288	0.9728	0.8642	0.8749	0.9072	0.6612	0.9416	0.7172	0.9044	0.6615	0.6187	0.6106	0.4636	0.5065	0.8535	
Phr29-Norm	0.4570	0.4925	0.8980	0.9703	0.9218	0.9348	0.8792	0.5117	0.9317	0.7410	0.8904	0.5119	0.4571	0.4154	0.2803	0.3396	0.8833	0.2129
Phr29-Real	0.5363	0.5266	0.8719	0.9783	0.8945	0.9125												

Table 33 – Oracle 02 – CPqD – Feature-based test oracles against normalization issues

	TAMM1	TAMM2	TAMM3	TAMM4	TAMM5	TAMM6	TAMM7	TAMM8	TAMM9	TAMM10	Euc. Dist.
Phr01-Norm	0,7895	0,0063	0,0077	0,0001	0,0001	0,0001	0,0000	0,0001	0,0001	0,0000	
Phr01-Real	0,7876	0,0189	0,0227	0,0004	0,0004	0,0005	0,0000	0,0004	0,0005	0,0000	0,0197
Phr02-Norm	0,8369	0,0357	0,0409	0,0013	0,0015	0,0017	0,0000	0,0015	0,0017	0,0001	
Phr02-Real	0,8264	0,0567	0,0647	0,0032	0,0037	0,0042	0,0002	0,0037	0,0042	0,0003	0,0338
Phr03-Norm	0,9343	0,0740	0,0828	0,0055	0,0061	0,0069	0,0004	0,0061	0,0068	0,0006	
Phr03-Real	0,9535	0,0888	0,0960	0,0079	0,0085	0,0092	0,0007	0,0085	0,0092	0,0009	0,0281
Phr04-Norm	0,9457	0,1058	0,1137	0,0112	0,0121	0,0129	0,0012	0,0119	0,0128	0,0015	
Phr04-Real	0,9668	0,1238	0,1331	0,0153	0,0165	0,0177	0,0019	0,0163	0,0176	0,0024	0,0353
Phr05-Norm	0,8961	0,1399	0,1496	0,0196	0,0209	0,0224	0,0027	0,0208	0,0222	0,0033	
Phr05-Real	0,9283	0,1534	0,1627	0,0235	0,0250	0,0265	0,0036	0,0248	0,0263	0,0043	0,0385
Phr06-Norm	0,9794	0,1660	0,1730	0,0276	0,0287	0,0299	0,0046	0,0286	0,0298	0,0052	
Phr06-Real	0,8095	0,1796	0,1835	0,0323	0,0329	0,0337	0,0058	0,0328	0,0335	0,0062	0,1710
Phr07-Norm	0,8813	0,1951	0,1983	0,0381	0,0387	0,0393	0,0074	0,0385	0,0392	0,0078	
Phr07-Real	0,8269	0,2114	0,2179	0,0447	0,0461	0,0475	0,0095	0,0459	0,0473	0,0104	
Phr08-Norm	0,8923	0,2313	0,2408	0,0535	0,0557	0,0580	0,0124	0,0556	0,0578	0,0140	
Phr08-Real	0,9329	0,2537	0,2647	0,0643	0,0672	0,0701	0,0163	0,0671	0,0700	0,0186	0,0586
Phr09-Norm	1,0000	0,2711	0,2839	0,0734	0,0769	0,0806	0,0199	0,0769	0,0805	0,0229	
Phr09-Real	0,9153	0,2835	0,2966	0,0804	0,0840	0,0879	0,0228	0,0840	0,0878	0,0261	0,0881
Phr10-Norm	0,9277	0,2983	0,3111	0,0890	0,0928	0,0968	0,0266	0,0927	0,0967	0,0301	
Phr10-Real	0,8717	0,3167	0,3315	0,1003	0,1049	0,1098	0,0318	0,1050	0,1098	0,0364	0,0687
Phr11-Norm	0,8803	0,3320	0,3487	0,1103	0,1157	0,1215	0,0366	0,1158	0,1215	0,0424	
Phr11-Real	0,8807	0,3441	0,3611	0,1183	0,1242	0,1303	0,0407	0,1243	0,1304	0,0470	0,0265
Phr12-Norm	0,8139	0,3597	0,3779	0,1294	0,1360	0,1429	0,0466	0,1361	0,1430	0,0540	
Phr12-Real	0,8153	0,3789	0,3958	0,1436	0,1500	0,1567	0,0544	0,1501	0,1568	0,0620	0,0424
Phr13-Norm	0,8507	0,3996	0,4173	0,1597	0,1667	0,1741	0,0638	0,1669	0,1742	0,0727	
Phr13-Real	0,9320	0,4205	0,4405	0,1769	0,1853	0,1941	0,0744	0,1854	0,1942	0,0855	0,0982
Phr14-Norm	0,9465	0,4366	0,4556	0,1904	0,1988	0,2075	0,0831	0,1989	0,2076	0,0945	
Phr14-Real	0,9459	0,4482	0,4664	0,2008	0,2089	0,2175	0,0900	0,2092	0,2177	0,1014	0,0294
Phr15-Norm	0,8382	0,4600	0,4779	0,2116	0,2198	0,2285	0,0974	0,2200	0,2286	0,1092	
Phr15-Real	0,7767	0,4728	0,4878	0,2234	0,2305	0,2379	0,1056	0,2306	0,2380	0,1160	0,0686
Phr16-Norm	0,8955	0,4868	0,4985	0,2369	0,2425	0,2484	0,1153	0,2426	0,2484	0,1238	
Phr16-Real	0,9050	0,5016	0,5122	0,2516	0,2569	0,2623	0,1263	0,2569	0,2623	0,1344	0,0418
Phr17-Norm	0,8597	0,5158	0,5257	0,2659	0,2710	0,2762	0,1371	0,2710	0,2762	0,1452	
Phr17-Real	0,9057	0,5290	0,5383	0,2798	0,2846	0,2896	0,1480	0,2846	0,2895	0,1558	0,0600
Phr18-Norm	0,8028	0,5467	0,5579	0,2989	0,3049	0,3112	0,1635	0,3049	0,3111	0,1736	
Phr18-Real	0,8218	0,5697	0,5829	0,3245	0,3320	0,3399	0,1849	0,3319	0,3396	0,1982	0,0795
Phr19-Norm	0,8890	0,5899	0,6006	0,3480	0,3542	0,3606	0,2054	0,3540	0,3603	0,2165	
Phr19-Real	0,8889	0,6066	0,6132	0,3681	0,3720	0,3761	0,2233	0,3718	0,3758	0,2306	0,0498
Phr20-Norm	0,9442	0,6236	0,6291	0,3888	0,3921	0,3957	0,2424	0,3919	0,3954	0,2489	
Phr20-Real	0,9137	0,6420	0,6466	0,4119	0,4150	0,4182	0,2645	0,4148	0,4179	0,2704	0,0715
Phr21-Norm	0,9448	0,6589	0,6632	0,4341	0,4368	0,4398	0,2861	0,4367	0,4395	0,2916	
Phr21-Real	0,9265	0,6731	0,6778	0,4530	0,4561	0,4593	0,3050	0,4559	0,4590	0,3112	0,0580
Phr22-Norm	0,8735	0,6917	0,6955	0,4784	0,4809	0,4836	0,3310	0,4809	0,4834	0,3363	
Phr22-Real	0,8672	0,7156	0,7163	0,5121	0,5125	0,5130	0,3666	0,5125	0,5129	0,3675	0,0903
Phr23-Norm	0,8000	0,7358	0,7346	0,5413	0,5403	0,5394	0,3982	0,5402	0,5393	0,3962	
Phr23-Real	0,8518	0,7512	0,7483	0,5645	0,5619	0,5599	0,4239	0,5620	0,5598	0,4190	0,0813
Phr24-Norm	0,8434	0,7716	0,7680	0,5953	0,5927	0,5897	0,4593	0,5927	0,5899	0,4531	
Phr24-Real	0,8116	0,7969	0,7921	0,6346	0,6312	0,6274	0,5061	0,6308	0,6272	0,4968	0,1167
Phr25-Norm	0,8977	0,8183	0,8136	0,6697	0,6656	0,6621	0,5479	0,6653	0,6617	0,5385	
Phr25-Real	0,9003	0,8355	0,8337	0,6978	0,6965	0,6949	0,5829	0,6959	0,6946	0,5795	0,0918
Phr26-Norm	0,8619	0,8532	0,8545	0,7275	0,7288	0,7301	0,6208	0,7287	0,7297	0,6238	
Phr26-Real	0,8622	0,8715	0,8777	0,7593	0,7648	0,7707	0,6621	0,7644	0,7697	0,6762	0,1103
Phr27-Norm	0,9004	0,8892	0,8976	0,7901	0,7982	0,8059	0,7028	0,7978	0,8054	0,7232	
Phr27-Real	0,8613	0,9059	0,9149	0,8208	0,8285	0,8367	0,7436	0,8283	0,8361	0,7654	0,1012
Phr28-Norm	0,8723	0,9233	0,9308	0,8527	0,8593	0,8660	0,7875	0,8589	0,8657	0,8059	
Phr28-Real	0,9472	0,9405	0,9463	0,8840	0,8896	0,8954	0,8313	0,8894	0,8947	0,8465	0,1194
Phr29-Norm	0,9720	0,9563	0,9615	0,9142	0,9194	0,9247	0,8740	0,9194	0,9243	0,8886	
Phr29-Real	0,9414	0,9717	0,9741	0,9439	0,9461	0,9487	0,9172	0,9459	0,9485	0,9238	0,0888
Phr30-Norm	0,8120	0,9861	0,9863	0,9725	0,9723	0,9726	0,9586	0,9724	0,9726	0,9595	
Phr30-Real	0,8748	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,1071

Source: Elaborated by the author.

Table 34 – Oracle 01 – Ivona – Feature-based test oracles against normalization issues

	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7	Ex 8	Ex 9	Ex 10	Ex 11	Ex 12	Ex 13	Ex 14	Ex 15	Ex 16	Ex 17	Enc. Dist.
Phr01-Real	0.836	0.851	0.793	0.990	0.859	0.873	0.983	0.834	0.903	0.780	0.817	0.835	0.836	0.820	0.835	0.820	0.935	0.099
Phr01-Norm	0.804	0.812	0.808	0.989	0.873	0.881	0.983	0.801	0.884	0.800	0.833	0.801	0.804	0.811	0.795	0.801	0.912	
Phr02-Real	0.836	0.847	0.820	0.989	0.866	0.889	0.971	0.844	0.929	0.775	0.978	0.845	0.836	0.813	0.752	0.840	0.939	0.128
Phr02-Norm	0.801	0.822	0.756	0.991	0.859	0.878	0.973	0.827	0.920	0.754	0.993	0.827	0.802	0.759	0.686	0.822	0.945	
Phr03-Real	0.576	0.601	0.807	0.991	0.920	0.924	0.971	0.571	0.952	0.837	0.868	0.571	0.576	0.601	0.465	0.561	0.886	0.225
Phr03-Norm	0.619	0.641	0.957	0.986	0.953	0.957	0.963	0.599	0.970	0.907	0.816	0.599	0.619	0.645	0.548	0.592	0.856	
Phr04-Real	0.623	0.685	0.828	0.997	0.912	0.919	0.975	0.647	0.915	0.834	0.846	0.647	0.623	0.606	0.622	0.734	0.872	0.068
Phr04-Norm	0.627	0.683	0.865	0.992	0.914	0.933	0.962	0.647	0.930	0.859	0.830	0.647	0.628	0.606	0.583	0.719	0.868	
Phr05-Real	0.653	0.672	0.931	0.997	0.981	1.000	0.920	0.658	0.980	0.986	0.810	0.658	0.654	0.663	0.510	0.657	0.843	0.219
Phr05-Norm	0.655	0.692	0.925	0.992	0.916	0.929	0.955	0.660	0.941	0.867	0.823	0.660	0.656	0.681	0.646	0.685	0.882	
Phr06-Real	0.514	0.578	0.968	1.000	1.000	0.995	0.961	0.526	0.997	0.973	0.847	0.526	0.514	0.513	0.444	0.478	0.852	0.190
Phr06-Norm	0.524	0.595	1.000	0.998	0.990	0.986	0.962	0.527	0.999	1.000	0.942	0.800	0.862	0.859	0.754	0.828	0.860	0.458
Phr07-Real	0.859	0.859	0.923	0.993	0.941	0.969	0.928	0.862	1.000	0.942	0.800	0.862	0.859	0.865	0.754	0.828	0.860	
Phr07-Norm	1.000	1.000	0.928	0.999	0.932	0.962	0.961	1.000	0.995	0.960	0.830	1.000	1.000	1.000	1.000	1.000	0.874	
Phr08-Real	0.799	0.810	0.857	0.994	0.945	0.962	0.947	0.810	0.952	0.895	0.805	0.810	0.799	0.759	0.650	0.737	0.884	0.081
Phr08-Norm	0.774	0.794	0.872	0.971	0.958	0.972	0.932	0.798	0.964	0.895	0.792	0.798	0.775	0.742	0.628	0.775	0.851	
Phr09-Real	0.724	0.746	0.804	0.982	0.863	0.892	0.947	0.754	0.962	0.790	0.852	0.754	0.725	0.698	0.549	0.632	0.881	0.507
Phr09-Norm	0.865	0.871	0.837	0.986	0.900	0.913	0.969	0.876	0.981	0.848	0.830	0.876	0.865	0.880	0.843	0.842	0.912	
Phr10-Real	0.670	0.687	0.819	0.992	0.920	0.930	0.951	0.673	0.986	0.833	0.839	0.673	0.670	0.672	0.614	0.673	0.885	0.257
Phr10-Norm	0.775	0.771	0.831	0.996	0.913	0.931	0.962	0.771	0.977	0.834	0.800	0.772	0.775	0.779	0.653	0.721	0.875	
Phr11-Real	0.680	0.749	0.708	0.991	0.812	0.826	1.000	0.717	0.919	0.678	0.862	0.717	0.680	0.656	0.746	0.930	0.938	0.184
Phr11-Norm	0.750	0.809	0.723	0.986	0.822	0.842	0.997	0.777	0.911	0.711	0.824	0.778	0.750	0.744	0.743	0.880	0.936	
Phr12-Real	0.656	0.672	0.830	0.991	0.886	0.899	0.963	0.662	0.891	0.773	0.830	0.662	0.656	0.676	0.560	0.600	0.903	0.062
Phr12-Norm	0.679	0.685	0.837	0.986	0.879	0.892	0.965	0.679	0.894	0.769	0.838	0.679	0.679	0.696	0.585	0.626	0.908	
Phr13-Real	0.692	0.727	0.807	0.963	0.870	0.892	0.974	0.714	0.940	0.774	0.821	0.714	0.692	0.683	0.617	0.719	0.878	0.183
Phr13-Norm	0.653	0.689	0.877	0.998	0.926	0.935	0.964	0.668	0.939	0.838	0.808	0.668	0.653	0.649	0.551	0.662	0.861	
Phr14-Real	0.635	0.740	0.909	0.959	0.899	0.912	0.970	0.645	0.968	0.844	0.805	0.645	0.635	0.582	0.690	0.782	0.890	0.321
Phr14-Norm	0.538	0.604	0.924	0.956	0.907	0.916	0.976	0.555	0.932	0.843	0.790	0.556	0.538	0.528	0.519	0.658	0.873	
Phr15-Real	0.437	0.530	0.976	0.972	0.941	0.941	0.942	0.461	0.950	0.863	0.823	0.461	0.437	0.418	0.388	0.480	0.869	0.666
Phr15-Norm	0.676	0.694	0.942	0.964	0.902	0.907	0.971	0.664	0.928	0.852	0.906	0.664	0.676	0.691	0.706	0.638	0.956	
Phr16-Real	0.589	0.630	0.842	0.961	0.904	0.914	0.956	0.598	0.964	0.824	0.967	0.598	0.588	0.583	0.578	0.680	0.938	0.138
Phr16-Norm	0.645	0.683	0.848	0.967	0.905	0.919	0.955	0.650	0.960	0.846	0.978	0.650	0.645	0.631	0.613	0.683	0.953	
Phr17-Real	0.606	0.661	0.901	0.984	0.877	0.902	0.978	0.620	0.991	0.812	0.998	0.620	0.606	0.580	0.533	0.608	0.904	0.180
Phr17-Norm	0.660	0.717	0.895	0.957	0.883	0.900	0.971	0.668	0.995	0.818	1.000	0.668	0.660	0.659	0.622	0.669	0.917	
Phr18-Real	0.788	0.800	0.828	0.991	0.897	0.919	0.967	0.800	0.927	0.818	0.909	0.800	0.788	0.766	0.675	0.769	0.906	0.282
Phr18-Norm	0.880	0.865	0.858	0.965	0.917	0.935	0.958	0.885	0.927	0.850	0.795	0.885	0.880	0.868	0.780	0.843	0.873	
Phr19-Real	0.462	0.523	0.888	0.994	0.915	0.902	0.987	0.467	0.945	0.814	0.976	0.467	0.461	0.470	0.340	0.408	0.917	0.267
Phr19-Norm	0.549	0.595	0.848	0.970	0.905	0.901	0.980	0.541	0.945	0.826	0.984	0.541	0.549	0.566	0.480	0.497	0.935	
Phr20-Real	0.694	0.715	0.913	0.998	0.931	0.935	0.968	0.695	0.986	0.854	0.978	0.695	0.694	0.696	0.606	0.665	0.921	0.126
Phr20-Norm	0.696	0.732	0.993	0.964	0.950	0.958	0.967	0.696	0.974	0.907	0.965	0.696	0.696	0.672	0.620	0.712	0.894	
Phr21-Real	0.703	0.741	0.753	0.991	0.797	0.816	0.989	0.702	0.940	0.690	0.992	0.702	0.703	0.715	0.681	0.698	0.966	0.139
Phr21-Norm	0.650	0.693	0.780	0.976	0.799	0.811	0.996	0.646	0.917	0.695	0.973	0.646	0.650	0.677	0.673	0.658	0.973	
Phr22-Real	0.556	0.602	0.879	0.982	0.899	0.911	0.979	0.567	0.924	0.811	0.837	0.567	0.557	0.555	0.443	0.527	0.847	0.111
Phr22-Norm	0.562	0.630	0.871	0.952	0.909	0.913	0.976	0.590	0.918	0.812	0.836	0.590	0.562	0.547	0.502	0.600	0.868	
Phr23-Real	0.578	0.654	0.821	0.962	0.892	0.900	0.950	0.600	0.982	0.781	0.863	0.600	0.578	0.571	0.561	0.678	0.898	0.149
Phr23-Norm	0.623	0.675	0.896	0.966	0.873	0.886	0.972	0.629	0.943	0.753	0.863	0.629	0.623	0.640	0.607	0.687	0.904	
Phr24-Real	0.509	0.583	0.855	0.965	0.912	0.923	0.971	0.537	0.930	0.823	0.837	0.537	0.510	0.497	0.426	0.520	0.851	0.088
Phr24-Norm	0.520	0.579	0.919	0.970	0.931	0.939	0.973	0.540	0.928	0.848	0.852	0.540	0.520	0.511	0.421	0.560	0.857	
Phr25-Real	0.845	0.837	0.932	0.969	0.934	0.962	0.951	0.855	0.999	0.919	0.789	0.855	0.845	0.839	0.785	0.838	0.815	0.235
Phr25-Norm	0.931	0.920	0.924	0.959	0.905	0.925	0.972	0.930	0.971	0.870	0.817	0.930	0.931	0.922	0.845	0.855	0.880	
Phr26-Real	0.916	0.913	0.809	0.966	0.893	0.923	0.937	0.916	0.963	0.837	0.794	0.916	0.916	0.927	0.777	0.854	0.877	0.254
Phr26-Norm	0.841	0.830	0.827	0.965	0.890	0.923	0.943	0.837	0.953	0.845	0.823	0.837	0.841	0.831	0.659	0.749	0.897	
Phr27-Real	0.693	0.741	0.836	0.971	0.910	0.923	0.951	0.711	0.959	0.836	0.939	0.711	0.694	0.663	0.625	0.719	0.918	0.084
Phr27-Norm	0.720	0.754	0.846	0.967	0.910	0.923	0.957	0.745	0.958	0.836	0.947	0.745	0.720	0.710	0.651	0.718	0.913	
Phr28-Real	0.634	0.673	0.889	0.962	0.973	0.985	0.934	0.628	0.978	0.971	0.795	0.629	0.634	0.622	0.573	0.628	0.851	0.127
Phr28-Norm	0.665	0.670	0.923	0.965	0.944	0.958	0.928	0.651	0.975	0.922	0.837	0.651	0.665	0.691	0.593	0.650	0.877	
Phr29-Real	0.487	0.550	0.849	0.951	0.899	0.909	0.960	0.491	0.947	0.799	0.848	0.492	0.487	0.494	0.445	0.489	0.877	0.063
Phr29-Norm	0.478	0.543	0.887	0.959	0.918	0.923	0.969	0.482	0.943	0.826	0.853	0.482	0.478	0.484	0.423	0.501	0.877	
Phr30-Real	0.864	0.898	0.512	0.972	0.712	0.728	0.949	0.875	0.952	0.550	0.885	0.875	0.864	0.831	0.943	0.932	1.000	0.761
Phr30-Norm	0.675	0.716	0.787	0.965	0.900	0.909	0.955	0.685	0.941	0.835	0.846	0.685	0.675	0.666	0.618	0.745	0.912	

Source: Elaborated by the author.

Table 35 – Oracle 02 – Ivona – Feature-based test oracles against normalization issues

	TAMM1	TAMM2	TAMM3	TAMM4	TAMM5	TAMM6	TAMM7	TAMM8	TAMM9	TAMM10	Euc. Dist
Phr01-Real	0.877	0.821	0.877	0.674	0.720	0.768	0.553	0.719	0.767	0.673	0,012
Phr01-Norm	0.881	0.823	0.879	0.678	0.723	0.772	0.559	0.723	0.771	0.678	
Phr02-Real	0.909	0.827	0.881	0.684	0.728	0.776	0.565	0.727	0.775	0.684	0,019
Phr02-Norm	0.903	0.831	0.884	0.690	0.734	0.782	0.573	0.733	0.781	0.691	
Phr03-Real	0.921	0.834	0.887	0.696	0.739	0.786	0.580	0.738	0.785	0.697	0,028
Phr03-Norm	0.947	0.837	0.888	0.700	0.743	0.789	0.586	0.742	0.788	0.701	
Phr04-Real	0.921	0.840	0.890	0.705	0.748	0.792	0.592	0.747	0.791	0.705	0,018
Phr04-Norm	0.930	0.843	0.893	0.711	0.753	0.796	0.600	0.752	0.796	0.711	
Phr05-Real	0.997	0.846	0.894	0.716	0.757	0.800	0.606	0.756	0.799	0.715	0,075
Phr05-Norm	0.923	0.849	0.896	0.720	0.760	0.803	0.611	0.759	0.802	0.719	
Phr06-Real	0.966	0.851	0.898	0.725	0.764	0.806	0.617	0.763	0.805	0.723	0,024
Phr06-Norm	0.944	0.854	0.899	0.729	0.767	0.808	0.622	0.766	0.807	0.726	
Phr07-Real	0.985	0.856	0.901	0.733	0.771	0.811	0.628	0.770	0.810	0.731	0,015
Phr07-Norm	0.986	0.860	0.903	0.739	0.776	0.816	0.635	0.776	0.815	0.737	
Phr08-Real	0.987	0.863	0.906	0.745	0.782	0.821	0.642	0.781	0.820	0.744	0,023
Phr08-Norm	1.000	0.867	0.909	0.751	0.788	0.826	0.651	0.787	0.825	0.751	
Phr09-Real	0.900	0.870	0.911	0.756	0.793	0.831	0.658	0.792	0.830	0.757	0,036
Phr09-Norm	0.935	0.872	0.913	0.760	0.796	0.834	0.662	0.795	0.833	0.761	
Phr10-Real	0.942	0.874	0.915	0.765	0.800	0.837	0.669	0.799	0.836	0.766	0,019
Phr10-Norm	0.954	0.878	0.917	0.770	0.805	0.841	0.676	0.804	0.840	0.772	
Phr11-Real	0.851	0.880	0.919	0.775	0.809	0.845	0.683	0.809	0.844	0.777	0,022
Phr11-Norm	0.870	0.883	0.921	0.779	0.813	0.848	0.688	0.812	0.847	0.781	
Phr12-Real	0.883	0.886	0.923	0.784	0.817	0.852	0.694	0.817	0.852	0.787	0,018
Phr12-Norm	0.875	0.889	0.925	0.791	0.823	0.857	0.703	0.822	0.856	0.793	
Phr13-Real	0.916	0.893	0.928	0.797	0.828	0.861	0.712	0.828	0.860	0.799	0,028
Phr13-Norm	0.939	0.896	0.930	0.804	0.834	0.866	0.720	0.833	0.865	0.805	
Phr14-Real	0.911	0.899	0.932	0.809	0.838	0.868	0.727	0.838	0.868	0.810	0,010
Phr14-Norm	0.909	0.901	0.933	0.813	0.841	0.871	0.733	0.841	0.871	0.813	
Phr15-Real	0.925	0.904	0.935	0.817	0.845	0.874	0.738	0.844	0.873	0.817	0,044
Phr15-Norm	0.883	0.906	0.936	0.821	0.848	0.877	0.744	0.848	0.876	0.821	
Phr16-Real	0.904	0.909	0.938	0.826	0.852	0.880	0.750	0.852	0.879	0.825	0,014
Phr16-Norm	0.910	0.911	0.940	0.831	0.857	0.883	0.757	0.856	0.883	0.830	
Phr17-Real	0.904	0.914	0.942	0.835	0.861	0.886	0.764	0.860	0.886	0.835	0,012
Phr17-Norm	0.909	0.916	0.943	0.840	0.864	0.890	0.770	0.864	0.889	0.839	
Phr18-Real	0.944	0.919	0.946	0.846	0.869	0.894	0.777	0.869	0.894	0.846	0,030
Phr18-Norm	0.966	0.923	0.949	0.852	0.876	0.900	0.787	0.876	0.900	0.854	
Phr19-Real	0.877	0.927	0.951	0.859	0.881	0.904	0.796	0.881	0.904	0.860	0,016
Phr19-Norm	0.868	0.930	0.953	0.864	0.886	0.908	0.804	0.885	0.907	0.864	
Phr20-Real	0.930	0.933	0.955	0.870	0.891	0.912	0.812	0.890	0.911	0.870	0,027
Phr20-Norm	0.951	0.936	0.957	0.877	0.896	0.916	0.821	0.896	0.916	0.877	
Phr21-Real	0.814	0.939	0.959	0.882	0.901	0.920	0.828	0.901	0.920	0.883	0,014
Phr21-Norm	0.809	0.942	0.961	0.887	0.905	0.924	0.836	0.905	0.923	0.888	
Phr22-Real	0.885	0.945	0.963	0.894	0.911	0.928	0.845	0.910	0.927	0.894	0,026
Phr22-Norm	0.902	0.950	0.966	0.902	0.917	0.933	0.856	0.917	0.932	0.901	
Phr23-Real	0.897	0.953	0.968	0.908	0.922	0.937	0.866	0.922	0.936	0.906	0,022
Phr23-Norm	0.878	0.956	0.970	0.913	0.926	0.940	0.873	0.926	0.940	0.911	
Phr24-Real	0.918	0.959	0.972	0.920	0.932	0.944	0.882	0.931	0.943	0.917	0,024
Phr24-Norm	0.932	0.963	0.974	0.928	0.938	0.948	0.894	0.938	0.948	0.924	
Phr25-Real	0.980	0.967	0.976	0.935	0.944	0.953	0.904	0.944	0.953	0.931	0,046
Phr25-Norm	0.937	0.970	0.979	0.941	0.950	0.958	0.914	0.950	0.958	0.938	
Phr26-Real	0.952	0.974	0.982	0.948	0.956	0.964	0.923	0.956	0.963	0.946	0,021
Phr26-Norm	0.942	0.977	0.985	0.955	0.962	0.969	0.933	0.962	0.969	0.954	
Phr27-Real	0.935	0.980	0.987	0.961	0.967	0.974	0.942	0.967	0.974	0.961	0,018
Phr27-Norm	0.933	0.984	0.989	0.967	0.973	0.979	0.951	0.973	0.979	0.968	
Phr28-Real	0.975	0.987	0.992	0.973	0.978	0.983	0.960	0.978	0.983	0.975	0,031
Phr28-Norm	0.948	0.990	0.994	0.979	0.983	0.987	0.969	0.983	0.987	0.980	
Phr29-Real	0.866	0.992	0.995	0.985	0.988	0.991	0.977	0.988	0.991	0.986	0,019
Phr29-Norm	0.880	0.995	0.997	0.991	0.992	0.993	0.986	0.992	0.993	0.990	
Phr30-Real	0.721	0.998	0.998	0.995	0.996	0.997	0.993	0.996	0.997	0.995	0,187
Phr30-Norm	0.907	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	

Table 36 – Oracle 01 – ODDCAST – Feature-based test oracles against normalization issues

	Ext 1	Ext 2	Ext 3	Ext 4	Ext 5	Ext 6	Ext 7	Ext 8	Ext 9	Ext 10	Ext 11	Ext 12	Ext 13	Ext 14	Ext 15	Ext 16	Ext 17	Eucl. Dist
Phr01-Real	1.000	1.000	0.761	0.970	0.832	0.859	0.976	1.000	0.943	0.743	0.832	1.000	1.000	0.866	0.802	0.898	0.891	0.417
Phr01-Norm	0.857	0.860	0.808	0.985	0.872	0.896	0.956	0.870	0.955	0.789	0.819	0.869	0.857	0.866	0.802	0.898	0.891	0.246
Phr02-Real	0.796	0.867	0.713	0.981	0.812	0.840	0.984	0.816	0.924	0.696	0.979	0.816	0.796	0.765	0.747	0.901	0.985	0.207
Phr02-Norm	0.716	0.791	0.766	0.975	0.831	0.851	0.986	0.742	0.928	0.704	0.969	0.742	0.716	0.680	0.651	0.799	0.971	0.312
Phr03-Real	0.643	0.682	0.746	0.979	0.878	0.901	0.968	0.646	0.956	0.801	0.856	0.646	0.643	0.683	0.590	0.591	0.927	0.207
Phr03-Norm	0.583	0.628	0.743	0.991	0.913	0.932	0.959	0.587	0.954	0.849	0.840	0.587	0.583	0.606	0.483	0.539	0.908	0.148
Phr04-Real	0.756	0.813	0.757	0.979	0.887	0.911	0.976	0.779	0.966	0.811	0.824	0.779	0.756	0.749	0.706	0.829	0.891	0.312
Phr04-Norm	0.645	0.728	0.797	0.980	0.916	0.942	0.963	0.671	0.944	0.859	0.818	0.671	0.645	0.629	0.595	0.736	0.863	0.148
Phr05-Real	0.663	0.689	0.699	0.984	0.863	0.889	0.951	0.655	0.942	0.786	0.818	0.655	0.663	0.686	0.573	0.632	0.922	0.148
Phr05-Norm	0.677	0.716	0.740	0.958	0.868	0.884	0.969	0.690	0.957	0.741	0.819	0.690	0.677	0.690	0.603	0.733	0.876	0.530
Phr06-Real	0.670	0.726	0.917	0.998	0.937	0.945	0.966	0.669	0.973	0.884	0.831	0.669	0.670	0.669	0.651	0.623	0.872	0.530
Phr06-Norm	0.494	0.590	1.000	0.972	1.000	1.000	0.950	0.500	0.989	1.000	0.806	0.500	0.494	0.490	0.439	0.433	0.840	0.368
Phr07-Real	0.950	0.958	0.768	0.975	0.880	0.912	0.961	0.965	0.964	0.852	0.824	0.966	0.949	0.941	0.883	0.922	0.901	0.368
Phr07-Norm	0.821	0.850	0.815	0.961	0.910	0.942	0.958	0.838	0.994	0.908	0.810	0.838	0.821	0.827	0.724	0.817	0.871	0.071
Phr08-Real	0.868	0.893	0.822	0.988	0.900	0.930	0.961	0.894	0.976	0.850	0.800	0.894	0.868	0.852	0.737	0.881	0.871	0.071
Phr08-Norm	0.860	0.891	0.808	0.981	0.894	0.920	0.969	0.882	0.963	0.833	0.812	0.882	0.860	0.832	0.738	0.906	0.887	0.195
Phr09-Real	0.780	0.847	0.807	0.987	0.906	0.932	0.976	0.798	0.978	0.873	0.806	0.798	0.780	0.742	0.789	0.879	0.884	0.195
Phr09-Norm	0.856	0.894	0.748	0.976	0.898	0.918	0.958	0.849	0.973	0.879	0.833	0.849	0.856	0.845	0.817	0.913	0.923	0.292
Phr10-Real	0.756	0.789	0.739	0.988	0.862	0.883	0.970	0.762	0.964	0.746	0.857	0.762	0.756	0.759	0.715	0.762	0.911	0.292
Phr10-Norm	0.855	0.874	0.797	0.953	0.883	0.903	0.973	0.858	0.983	0.792	0.816	0.858	0.855	0.864	0.806	0.861	0.881	0.331
Phr11-Real	0.819	0.870	0.712	0.985	0.805	0.840	1.000	0.815	0.947	0.725	0.833	0.815	0.819	0.788	0.761	0.788	0.985	0.331
Phr11-Norm	0.704	0.783	0.681	0.958	0.818	0.849	0.983	0.717	0.953	0.724	0.821	0.717	0.704	0.659	0.599	0.690	0.949	0.174
Phr12-Real	0.693	0.731	0.792	0.984	0.857	0.874	0.972	0.705	0.952	0.732	0.842	0.705	0.693	0.698	0.608	0.713	0.918	0.174
Phr12-Norm	0.750	0.773	0.762	0.955	0.838	0.864	0.981	0.751	0.955	0.724	0.831	0.751	0.750	0.758	0.702	0.769	0.922	0.138
Phr13-Real	0.800	0.833	0.702	0.977	0.866	0.900	0.973	0.825	0.937	0.777	0.824	0.825	0.800	0.783	0.685	0.856	0.894	0.138
Phr13-Norm	0.829	0.870	0.748	0.959	0.888	0.920	0.949	0.852	0.972	0.813	0.802	0.829	0.803	0.749	0.911	0.876	0.876	0.113
Phr14-Real	0.647	0.735	0.881	0.984	0.911	0.935	0.960	0.672	1.000	0.861	0.823	0.673	0.646	0.607	0.533	0.621	0.873	0.113
Phr14-Norm	0.607	0.706	0.875	0.987	0.906	0.930	0.961	0.636	1.000	0.852	0.828	0.636	0.606	0.571	0.474	0.592	0.884	0.044
Phr15-Real	0.556	0.635	0.853	1.000	0.905	0.915	0.954	0.574	0.954	0.829	0.848	0.574	0.555	0.547	0.495	0.573	0.910	0.044
Phr15-Norm	0.550	0.629	0.869	0.997	0.896	0.908	0.958	0.561	0.952	0.817	0.856	0.561	0.550	0.545	0.476	0.551	0.911	0.264
Phr16-Real	0.646	0.720	0.758	0.959	0.853	0.873	0.984	0.658	0.965	0.761	0.973	0.658	0.646	0.630	0.608	0.675	0.993	0.264
Phr16-Norm	0.753	0.817	0.745	0.987	0.862	0.880	0.968	0.758	0.957	0.778	0.959	0.758	0.753	0.727	0.671	0.719	0.983	0.254
Phr17-Real	0.768	0.816	0.746	0.945	0.870	0.891	0.969	0.758	0.989	0.806	1.000	0.758	0.767	0.747	0.770	0.766	1.000	0.254
Phr17-Norm	0.690	0.744	0.760	0.973	0.877	0.903	0.952	0.694	0.989	0.814	0.987	0.694	0.690	0.681	0.619	0.673	0.959	0.118
Phr18-Real	0.817	0.841	0.819	0.958	0.867	0.895	0.969	0.831	0.956	0.768	0.911	0.831	0.817	0.806	0.723	0.857	0.941	0.118
Phr18-Norm	0.814	0.839	0.838	0.985	0.881	0.903	0.954	0.826	0.964	0.790	0.927	0.826	0.814	0.806	0.758	0.852	0.954	0.104
Phr19-Real	0.600	0.649	0.791	0.967	0.887	0.891	0.973	0.598	0.942	0.788	0.927	0.598	0.600	0.613	0.548	0.605	0.957	0.104
Phr19-Norm	0.635	0.675	0.769	0.986	0.876	0.888	0.971	0.628	0.944	0.780	0.931	0.629	0.635	0.657	0.600	0.618	0.955	0.300
Phr20-Real	0.679	0.725	0.822	0.957	0.896	0.911	0.968	0.692	0.957	0.797	0.934	0.692	0.679	0.676	0.581	0.667	0.930	0.300
Phr20-Norm	0.781	0.820	0.865	0.983	0.901	0.923	0.966	0.783	0.977	0.853	0.925	0.784	0.781	0.778	0.703	0.768	0.964	0.105
Phr21-Real	0.767	0.807	0.695	0.945	0.820	0.846	0.976	0.767	0.958	0.751	0.975	0.767	0.767	0.775	0.742	0.834	0.980	0.105
Phr21-Norm	0.757	0.797	0.763	0.980	0.825	0.847	0.972	0.741	0.952	0.759	0.973	0.740	0.757	0.760	0.725	0.786	1.000	0.181
Phr22-Real	0.503	0.589	0.761	0.952	0.894	0.918	0.971	0.532	0.941	0.804	0.842	0.532	0.503	0.490	0.376	0.525	0.861	0.181
Phr22-Norm	0.554	0.647	0.746	0.976	0.884	0.906	0.973	0.584	0.932	0.787	0.859	0.584	0.554	0.533	0.470	0.601	0.879	0.271
Phr23-Real	0.608	0.689	0.719	0.979	0.857	0.874	0.952	0.629	0.951	0.725	0.824	0.629	0.608	0.606	0.571	0.667	0.924	0.271
Phr23-Norm	0.698	0.777	0.727	0.985	0.862	0.880	0.959	0.705	0.950	0.758	0.852	0.705	0.698	0.681	0.717	0.759	0.943	0.096
Phr24-Real	0.633	0.699	0.793	0.986	0.886	0.904	0.967	0.643	0.951	0.781	0.851	0.643	0.633	0.626	0.584	0.677	0.886	0.086
Phr24-Norm	0.615	0.678	0.804	0.984	0.883	0.900	0.969	0.629	0.953	0.779	0.845	0.629	0.615	0.615	0.524	0.633	0.891	0.051
Phr25-Real	0.841	0.846	0.803	0.946	0.918	0.948	0.954	0.843	0.987	0.900	0.816	0.843	0.841	0.847	0.764	0.834	0.880	0.104
Phr25-Norm	0.839	0.840	0.786	0.979	0.910	0.938	0.952	0.842	0.983	0.879	0.817	0.839	0.856	0.845	0.754	0.868	0.917	0.104
Phr26-Real	0.856	0.874	0.732	0.950	0.837	0.871	0.960	0.864	0.961	0.751	0.839	0.864	0.856	0.859	0.789	0.940	0.908	0.096
Phr26-Norm	0.866	0.890	0.759	0.980	0.856	0.884	0.960	0.884	0.946	0.769	0.855	0.884	0.866	0.859	0.789	0.940	0.908	0.096
Phr27-Real	0.738	0.775	0.691	0.956	0.872	0.892	0.963	0.743	0.950	0.786	0.966	0.743	0.738	0.705	0.603	0.663	0.996	0.096
Phr27-Norm	0.708	0.764	0.753	0.982	0.870	0.891	0.969	0.725	0.951	0.789	0.958	0.725	0.708	0.698	0.586	0.700	0.981	0.232
Phr28-Real	0.673	0.706	0.875	0.962	0.940	0.964	0.928	0.675	0.984	0.924	0.800	0.675	0.672	0.674	0.565	0.570	0.840	0.232
Phr28-Norm	0.751	0.820	0.850	0.989	0.933	0.957	0.944	0.745	0.979	0.939	0.847	0.745	0.751	0.709	0.647	0.630	0.893	0.232
Phr29-Real	0.633	0.665	0.787	0.955	0.868	0.887	0.966	0.617	0.934	0.779	0.844	0.617	0.633	0.678	0.610	0.602	0.920	0.233
Phr29-Norm	0.557	0.621	0.856	0.981	0.878	0.892	0.967	0.555	0.936	0.802	0.895	0.555	0.557	0.579	0.501	0.552	0.928	0.233
Phr30-Real	0.710	0.754	0.768	0.954	0.887	0.908	0.939	0.725	0.969	0.821	0.838	0.725	0.710	0.694	0.654	0.771	0.900	0.221
Phr30-Norm	0.793	0.825	0.749	0.982	0.857	0.874	0.948	0.793	0.944	0.779	0.849	0.793	0.793	0.772	0.711	0.813	0.962	0.221

Source: Elaborated by the author.

Table 37 – Oracle 02 – ODDCAST – Feature-based test oracles against normalization issues

	TDA 1	TDA 2	TDA 3	TDA 4	TDA 5	TDA 6	TDA 7	TDA 8	TDA 9	TDA 10	Euc. Dist.
Phr01-Real	0,912	0,870	0,907	0,757	0,789	0,823	0,658	0,789	0,823	0,747	0,027
Phr01-Norm	0,938	0,872	0,909	0,760	0,792	0,826	0,662	0,792	0,825	0,751	
Phr02-Real	0,892	0,874	0,911	0,764	0,796	0,829	0,668	0,795	0,829	0,755	0,020
Phr02-Norm	0,907	0,877	0,913	0,769	0,801	0,833	0,674	0,800	0,833	0,761	
Phr03-Real	0,918	0,880	0,915	0,774	0,804	0,837	0,681	0,804	0,836	0,765	0,039
Phr03-Norm	0,956	0,882	0,916	0,777	0,807	0,839	0,685	0,807	0,838	0,768	
Phr04-Real	0,958	0,884	0,917	0,781	0,811	0,841	0,690	0,810	0,841	0,772	0,033
Phr04-Norm	0,989	0,886	0,919	0,785	0,814	0,845	0,696	0,814	0,844	0,776	
Phr05-Real	0,894	0,888	0,920	0,789	0,817	0,847	0,700	0,817	0,847	0,780	0,019
Phr05-Norm	0,911	0,890	0,921	0,792	0,820	0,849	0,705	0,819	0,849	0,782	
Phr06-Real	0,945	0,892	0,923	0,795	0,823	0,851	0,708	0,822	0,851	0,786	0,053
Phr06-Norm	0,997	0,893	0,924	0,798	0,825	0,853	0,713	0,825	0,853	0,788	
Phr07-Real	0,964	0,895	0,925	0,801	0,828	0,856	0,718	0,828	0,855	0,792	0,030
Phr07-Norm	0,992	0,898	0,927	0,806	0,832	0,859	0,723	0,831	0,859	0,797	
Phr08-Real	0,990	0,900	0,929	0,810	0,836	0,863	0,729	0,835	0,862	0,801	0,019
Phr08-Norm	0,979	0,903	0,931	0,815	0,841	0,867	0,736	0,840	0,867	0,808	
Phr09-Real	0,979	0,905	0,933	0,820	0,845	0,871	0,742	0,844	0,870	0,813	0,010
Phr09-Norm	0,973	0,907	0,935	0,822	0,848	0,873	0,746	0,847	0,873	0,816	
Phr10-Real	0,919	0,909	0,936	0,826	0,851	0,876	0,751	0,850	0,875	0,820	0,043
Phr10-Norm	0,961	0,911	0,938	0,830	0,854	0,879	0,757	0,854	0,879	0,825	
Phr11-Real	0,899	0,913	0,939	0,834	0,858	0,882	0,762	0,857	0,882	0,829	0,021
Phr11-Norm	0,919	0,915	0,940	0,837	0,860	0,885	0,766	0,860	0,884	0,832	
Phr12-Real	0,907	0,917	0,942	0,841	0,864	0,888	0,771	0,863	0,887	0,836	0,019
Phr12-Norm	0,893	0,919	0,944	0,845	0,868	0,891	0,777	0,867	0,891	0,841	
Phr13-Real	0,958	0,922	0,946	0,850	0,872	0,894	0,784	0,871	0,894	0,846	0,033
Phr13-Norm	0,988	0,925	0,948	0,855	0,876	0,898	0,790	0,876	0,898	0,851	
Phr14-Real	0,976	0,927	0,949	0,858	0,879	0,901	0,796	0,879	0,900	0,855	0,010
Phr14-Norm	0,969	0,928	0,950	0,861	0,882	0,902	0,799	0,881	0,902	0,857	
Phr15-Real	0,941	0,930	0,951	0,864	0,884	0,905	0,804	0,884	0,904	0,861	0,016
Phr15-Norm	0,927	0,931	0,952	0,868	0,887	0,907	0,808	0,886	0,906	0,863	
Phr16-Real	0,894	0,933	0,953	0,871	0,890	0,909	0,813	0,889	0,908	0,866	0,027
Phr16-Norm	0,919	0,935	0,955	0,875	0,893	0,911	0,818	0,893	0,911	0,870	
Phr17-Real	0,916	0,937	0,956	0,878	0,896	0,914	0,823	0,896	0,914	0,874	0,016
Phr17-Norm	0,930	0,939	0,957	0,882	0,899	0,917	0,828	0,899	0,916	0,878	
Phr18-Real	0,947	0,941	0,959	0,886	0,903	0,920	0,834	0,903	0,920	0,883	0,020
Phr18-Norm	0,959	0,944	0,962	0,892	0,908	0,925	0,842	0,908	0,925	0,890	
Phr19-Real	0,912	0,947	0,963	0,897	0,912	0,928	0,849	0,912	0,928	0,894	0,013
Phr19-Norm	0,903	0,949	0,965	0,901	0,916	0,931	0,855	0,916	0,931	0,898	
Phr20-Real	0,941	0,951	0,966	0,905	0,919	0,934	0,861	0,919	0,934	0,902	0,021
Phr20-Norm	0,958	0,954	0,968	0,910	0,923	0,937	0,868	0,923	0,937	0,907	
Phr21-Real	0,886	0,956	0,970	0,913	0,927	0,940	0,873	0,926	0,940	0,912	0,010
Phr21-Norm	0,887	0,958	0,971	0,917	0,930	0,943	0,879	0,930	0,943	0,916	
Phr22-Real	0,945	0,960	0,973	0,922	0,934	0,946	0,886	0,934	0,946	0,920	0,017
Phr22-Norm	0,937	0,964	0,975	0,929	0,939	0,950	0,895	0,939	0,950	0,925	
Phr23-Real	0,901	0,966	0,976	0,933	0,943	0,953	0,902	0,943	0,953	0,930	0,010
Phr23-Norm	0,902	0,968	0,977	0,937	0,946	0,955	0,907	0,946	0,955	0,934	
Phr24-Real	0,930	0,970	0,979	0,942	0,950	0,959	0,914	0,950	0,958	0,938	0,016
Phr24-Norm	0,933	0,974	0,981	0,948	0,955	0,962	0,923	0,955	0,962	0,944	
Phr25-Real	0,990	0,976	0,983	0,953	0,959	0,966	0,931	0,959	0,966	0,949	0,014
Phr25-Norm	0,982	0,979	0,985	0,958	0,963	0,969	0,937	0,963	0,969	0,954	
Phr26-Real	0,932	0,981	0,986	0,962	0,968	0,973	0,944	0,967	0,973	0,960	0,015
Phr26-Norm	0,936	0,984	0,988	0,967	0,972	0,977	0,952	0,972	0,977	0,966	
Phr27-Real	0,938	0,986	0,990	0,972	0,976	0,981	0,959	0,976	0,981	0,971	0,013
Phr27-Norm	0,938	0,988	0,992	0,977	0,980	0,984	0,965	0,980	0,984	0,977	
Phr28-Real	1,000	0,990	0,994	0,981	0,984	0,987	0,972	0,984	0,987	0,981	0,026
Phr28-Norm	0,976	0,992	0,995	0,985	0,987	0,990	0,978	0,987	0,990	0,985	
Phr29-Real	0,884	0,995	0,997	0,989	0,991	0,993	0,984	0,991	0,993	0,990	0,011
Phr29-Norm	0,890	0,997	0,998	0,993	0,994	0,995	0,989	0,994	0,995	0,993	
Phr30-Real	0,945	0,998	0,999	0,997	0,997	0,998	0,995	0,997	0,998	0,996	0,027
Phr30-Norm	0,919	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	

Source: Elaborated by the author.

Table 38 – Oracle 01 – VOCALWARE – Feature-based test oracles against normalization issues

	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7	Ex 8	Ex 9	Ex 10	Ex 11	Ex 12	Ex 13	Ex 14	Ex 15	Ex 16	Ex 17	Euc. Dist
Phr01-Norm	0.849	0.814	0.897	0.955	0.889	0.906	0.922	0.834	0.960	0.823	0.907	0.834	0.849	0.870	0.758	0.828	0.917	0.205
Phr01-Real	0.819	0.797	0.818	0.963	0.884	0.906	0.926	0.815	0.941	0.800	0.899	0.815	0.819	0.819	0.602	0.757	0.917	
Phr02-Norm	0.817	0.819	0.795	0.967	0.857	0.886	0.939	0.836	0.955	0.762	0.959	0.835	0.818	0.811	0.624	0.830	0.959	0.124
Phr02-Real	0.790	0.799	0.788	0.969	0.853	0.884	0.939	0.813	0.944	0.749	0.952	0.813	0.791	0.787	0.548	0.756	0.953	
Phr03-Norm	0.629	0.652	0.817	0.969	0.900	0.919	0.922	0.640	0.984	0.816	0.889	0.640	0.632	0.479	0.609	0.886	0.309	
Phr03-Real	0.711	0.721	0.791	0.969	0.870	0.895	0.933	0.702	0.983	0.787	0.893	0.702	0.711	0.723	0.681	0.737	0.920	
Phr04-Norm	0.668	0.720	0.918	0.971	0.916	0.935	0.929	0.679	0.965	0.849	0.930	0.679	0.668	0.656	0.560	0.699	0.894	0.122
Phr04-Real	0.714	0.728	0.900	0.972	0.914	0.943	0.915	0.722	0.969	0.861	0.911	0.714	0.706	0.506	0.697	0.874	0.908	
Phr05-Norm	0.702	0.709	0.810	0.958	0.843	0.871	0.950	0.702	0.962	0.742	0.927	0.702	0.702	0.710	0.483	0.583	0.908	0.240
Phr05-Real	0.793	0.780	0.823	0.951	0.838	0.875	0.946	0.798	0.978	0.752	0.913	0.797	0.793	0.809	0.566	0.577	0.902	
Phr06-Norm	0.561	0.638	1.000	0.995	1.000	1.000	0.914	0.565	0.997	1.000	0.914	0.565	0.560	0.582	0.552	0.621	0.912	0.173
Phr06-Real	0.628	0.658	0.995	0.971	0.970	0.982	0.924	0.613	0.999	0.945	0.920	0.613	0.628	0.550	0.552	0.602	0.884	
Phr07-Norm	1.000	0.934	0.768	0.983	0.875	0.911	0.911	0.999	0.976	0.856	0.901	0.999	1.000	1.000	0.666	0.778	0.917	0.360
Phr07-Real	0.848	0.817	0.802	0.965	0.863	0.897	0.921	0.850	0.966	0.814	0.911	0.849	0.848	0.861	0.628	0.791	0.921	
Phr08-Norm	0.918	0.891	0.828	0.985	0.875	0.901	0.930	0.927	0.914	0.798	0.900	0.927	0.918	0.908	0.723	0.830	0.927	0.137
Phr08-Real	0.877	0.862	0.831	0.957	0.886	0.916	0.917	0.899	0.961	0.826	0.896	0.898	0.878	0.867	0.648	0.811	0.897	
Phr09-Norm	1.000	1.000	0.692	0.988	0.841	0.876	0.973	1.000	0.983	0.780	0.919	1.000	1.000	0.979	1.000	1.000	0.957	0.610
Phr09-Real	0.792	0.804	0.727	0.963	0.831	0.855	1.000	0.828	0.984	0.721	0.909	0.828	0.792	0.789	0.683	0.798	0.892	
Phr10-Norm	0.791	0.784	0.842	0.986	0.880	0.900	0.943	0.811	0.953	0.785	0.900	0.811	0.791	0.781	0.515	0.636	0.918	0.127
Phr10-Real	0.753	0.750	0.803	0.965	0.882	0.911	0.931	0.765	0.988	0.796	0.904	0.765	0.753	0.752	0.460	0.651	0.899	
Phr11-Norm	0.726	0.748	0.874	0.996	0.859	0.888	0.941	0.737	0.980	0.781	0.914	0.737	0.727	0.725	0.543	0.684	0.917	0.306
Phr11-Real	0.825	0.814	0.813	0.970	0.847	0.880	0.939	0.828	1.000	0.763	0.919	0.828	0.824	0.832	0.715	0.770	0.918	
Phr12-Norm	0.683	0.681	0.859	0.990	0.837	0.856	0.941	0.677	0.926	0.704	0.940	0.677	0.683	0.710	0.531	0.605	0.959	0.118
Phr12-Real	0.699	0.694	0.871	0.954	0.869	0.889	0.915	0.699	0.924	0.773	0.904	0.699	0.699	0.724	0.509	0.614	0.923	
Phr13-Norm	0.830	0.816	0.838	1.000	0.884	0.916	0.921	0.831	0.966	0.808	0.890	0.831	0.830	0.839	0.649	0.776	0.908	0.247
Phr13-Real	0.782	0.776	0.775	0.962	0.831	0.874	0.944	0.801	0.976	0.801	0.931	0.801	0.782	0.794	0.495	0.683	0.911	
Phr14-Norm	0.626	0.683	0.920	0.994	0.866	0.892	0.935	0.654	0.954	0.778	0.944	0.654	0.626	0.627	0.511	0.617	0.910	0.288
Phr14-Real	0.686	0.736	0.922	0.968	0.858	0.887	0.931	0.708	0.968	0.782	0.938	0.708	0.686	0.695	0.696	0.783	0.911	
Phr15-Norm	0.661	0.683	0.835	0.981	0.897	0.915	0.926	0.662	0.973	0.805	0.962	0.661	0.678	0.583	0.632	0.632	0.940	0.201
Phr15-Real	0.596	0.639	0.875	0.965	0.911	0.923	0.905	0.613	0.969	0.812	0.940	0.613	0.596	0.613	0.467	0.570	0.913	
Phr16-Norm	0.819	0.797	0.844	0.994	0.875	0.888	0.935	0.802	0.957	0.777	0.979	0.802	0.820	0.848	0.704	0.714	0.991	0.394
Phr16-Real	0.712	0.780	0.750	0.960	0.847	0.870	0.942	0.715	0.960	0.749	0.985	0.715	0.712	0.650	0.489	0.577	0.967	
Phr17-Norm	0.750	0.768	0.862	0.994	0.875	0.904	0.943	0.750	0.971	0.814	0.942	0.751	0.750	0.748	0.684	0.716	0.949	0.419
Phr17-Real	0.851	0.878	0.885	0.959	0.860	0.879	0.933	0.848	0.977	0.780	0.956	0.847	0.851	0.825	0.945	0.929	0.970	
Phr18-Norm	0.866	0.826	0.876	0.994	0.893	0.912	0.945	0.869	0.948	0.805	0.920	0.868	0.866	0.880	0.653	0.800	0.911	0.383
Phr18-Real	0.960	0.936	0.766	0.962	0.861	0.884	0.939	0.970	0.944	0.758	0.948	0.970	0.944	0.936	0.858	0.976	0.960	
Phr19-Norm	0.644	0.673	0.900	0.992	0.923	0.929	0.925	0.628	0.960	0.826	0.956	0.628	0.644	0.662	0.620	0.672	0.932	0.306
Phr19-Real	0.550	0.610	0.933	0.972	0.930	0.938	0.924	0.549	0.958	0.834	0.935	0.548	0.550	0.560	0.439	0.557	0.906	
Phr20-Norm	0.736	0.755	0.939	1.000	0.938	0.965	0.925	0.742	0.972	0.898	0.930	0.742	0.736	0.734	0.488	0.600	0.889	0.681
Phr20-Real	0.922	0.917	0.827	0.965	0.883	0.911	0.928	0.903	0.984	0.822	0.942	0.903	0.922	0.893	0.902	0.902	0.941	
Phr21-Norm	0.746	0.768	0.771	0.992	0.797	0.825	0.948	0.731	0.946	0.705	0.995	0.732	0.746	0.762	0.801	0.828	1.000	0.088
Phr21-Real	0.729	0.763	0.745	0.962	0.813	0.843	0.948	0.728	0.929	0.734	0.976	0.728	0.729	0.744	0.756	0.844	0.970	
Phr22-Norm	0.587	0.632	0.798	0.987	0.851	0.871	0.941	0.611	0.915	0.730	0.975	0.611	0.586	0.589	0.357	0.542	0.932	0.233
Phr22-Real	0.642	0.691	0.776	0.967	0.847	0.870	0.948	0.658	0.937	0.733	0.998	0.668	0.642	0.629	0.496	0.665	0.943	
Phr23-Norm	0.640	0.674	0.833	0.987	0.872	0.891	0.956	0.656	0.976	0.753	0.934	0.656	0.640	0.656	0.484	0.608	0.904	0.604
Phr23-Real	0.816	0.818	0.798	0.962	0.837	0.862	0.965	0.819	0.988	0.731	0.943	0.818	0.816	0.826	0.828	0.881	0.956	
Phr24-Norm	0.616	0.646	0.871	0.996	0.894	0.914	0.924	0.629	0.942	0.811	0.926	0.629	0.629	0.635	0.414	0.577	0.905	0.079
Phr24-Real	0.620	0.658	0.865	0.969	0.891	0.913	0.922	0.629	0.942	0.790	0.912	0.628	0.620	0.633	0.426	0.518	0.893	
Phr25-Norm	0.935	0.881	0.864	0.991	0.899	0.928	0.934	0.916	0.985	0.890	0.910	0.916	0.935	0.956	0.787	0.837	0.895	0.453
Phr25-Real	0.765	0.762	0.893	0.969	0.920	0.946	0.910	0.764	0.988	0.889	0.899	0.764	0.765	0.778	0.614	0.688	0.873	
Phr26-Norm	0.844	0.817	0.815	0.988	0.863	0.894	0.923	0.848	0.950	0.797	0.936	0.848	0.843	0.846	0.508	0.731	0.956	0.507
Phr26-Real	0.986	0.929	0.806	0.954	0.862	0.898	0.924	0.980	0.967	0.803	0.919	0.979	0.986	0.981	0.840	0.929	0.955	
Phr27-Norm	0.772	0.796	0.864	0.988	0.902	0.918	0.944	0.789	0.973	0.838	1.000	0.789	0.772	0.760	0.698	0.755	0.975	0.293
Phr27-Real	0.698	0.748	0.914	0.967	0.901	0.924	0.934	0.707	0.965	0.832	0.981	0.707	0.697	0.676	0.524	0.627	0.941	
Phr28-Norm	0.761	0.752	0.944	0.994	0.893	0.914	0.944	0.751	0.974	0.831	0.944	0.752	0.761	0.777	0.696	0.744	0.911	0.535
Phr28-Real	0.591	0.619	0.932	0.962	0.916	0.936	0.935	0.592	0.970	0.841	0.922	0.591	0.591	0.605	0.412	0.531	0.874	
Phr29-Norm	0.655	0.691	0.932	0.994	0.873	0.890	0.928	0.647	0.944	0.802	0.955	0.647	0.655	0.685	0.702	0.727	0.942	0.221
Phr29-Real	0.704	0.701	0.850	0.962	0.875	0.902	0.922	0.693	0.963	0.807	0.959	0.693	0.704	0.719	0.568	0.617	0.952	
Phr30-Norm	0.892	0.859	0.780	0.980	0.838	0.859	0.953	0.881	0.952	0.753	0.936	0.881	0.892	0.933	0.976	0.979	0.979	0.215
Phr30-Real	0.866	0.842	0.759	0.963	0.854	0.887	0.931	0.850	0.959	0.788	0.911	0.850	0.866	0.893	0.900	0.808	0.930	

Source: Elaborated by the author.

Table 39 – Oracle 02 – VOCALWARE – Feature-based test oracles against normalization issues

	TDA 1	TDA 2	TDA 3	TDA 4	TDA 5	TDA 6	TDA 7	TDA 8	TDA 9	TDA 10	Euc. Dist.
Phr01-Norm	0,921	0,894	0,929	0,798	0,830	0,862	0,714	0,830	0,862	0,801	0,010
Phr01-Real	0,928	0,896	0,930	0,801	0,832	0,864	0,717	0,832	0,864	0,803	
Phr02-Norm	0,922	0,897	0,931	0,804	0,835	0,867	0,721	0,835	0,866	0,807	0,011
Phr02-Real	0,921	0,900	0,932	0,808	0,839	0,870	0,727	0,838	0,869	0,811	
Phr03-Norm	0,955	0,902	0,934	0,811	0,842	0,872	0,731	0,841	0,872	0,814	0,039
Phr03-Real	0,917	0,903	0,935	0,815	0,844	0,874	0,736	0,844	0,874	0,817	
Phr04-Norm	0,955	0,905	0,936	0,819	0,847	0,876	0,741	0,847	0,876	0,820	0,020
Phr04-Real	0,973	0,907	0,937	0,822	0,850	0,879	0,746	0,850	0,879	0,824	
Phr05-Norm	0,889	0,909	0,938	0,826	0,853	0,881	0,750	0,853	0,881	0,827	0,006
Phr05-Real	0,889	0,910	0,939	0,828	0,855	0,882	0,754	0,855	0,882	0,829	
Phr06-Norm	1,000	0,912	0,940	0,830	0,857	0,884	0,757	0,857	0,884	0,832	0,030
Phr06-Real	0,970	0,913	0,941	0,833	0,859	0,886	0,760	0,859	0,885	0,833	
Phr07-Norm	0,942	0,914	0,942	0,836	0,862	0,888	0,764	0,861	0,887	0,837	0,012
Phr07-Real	0,933	0,916	0,944	0,839	0,865	0,890	0,769	0,864	0,890	0,840	
Phr08-Norm	0,938	0,919	0,945	0,843	0,868	0,893	0,774	0,868	0,893	0,844	0,026
Phr08-Real	0,962	0,921	0,947	0,847	0,872	0,896	0,780	0,871	0,896	0,849	
Phr09-Norm	0,927	0,923	0,948	0,850	0,875	0,899	0,784	0,874	0,899	0,853	0,033
Phr09-Real	0,895	0,924	0,949	0,852	0,877	0,901	0,787	0,876	0,901	0,855	
Phr10-Norm	0,937	0,925	0,950	0,855	0,879	0,903	0,791	0,879	0,902	0,858	0,013
Phr10-Real	0,946	0,927	0,951	0,859	0,882	0,905	0,796	0,882	0,905	0,861	
Phr11-Norm	0,914	0,929	0,952	0,862	0,885	0,907	0,800	0,884	0,907	0,864	0,008
Phr11-Real	0,910	0,930	0,954	0,865	0,887	0,909	0,804	0,887	0,909	0,867	
Phr12-Norm	0,854	0,932	0,955	0,868	0,890	0,912	0,809	0,889	0,911	0,870	0,040
Phr12-Real	0,893	0,934	0,956	0,872	0,893	0,914	0,814	0,892	0,914	0,874	
Phr13-Norm	0,959	0,936	0,957	0,875	0,896	0,917	0,819	0,896	0,917	0,878	0,043
Phr13-Real	0,917	0,938	0,959	0,879	0,900	0,919	0,825	0,899	0,919	0,882	
Phr14-Norm	0,908	0,940	0,960	0,883	0,902	0,921	0,830	0,902	0,921	0,885	0,007
Phr14-Real	0,905	0,941	0,961	0,885	0,904	0,923	0,833	0,904	0,923	0,887	
Phr15-Norm	0,921	0,942	0,962	0,888	0,906	0,925	0,837	0,906	0,925	0,890	0,018
Phr15-Real	0,938	0,944	0,962	0,891	0,909	0,926	0,841	0,908	0,926	0,892	
Phr16-Norm	0,913	0,946	0,963	0,893	0,911	0,928	0,844	0,910	0,928	0,894	0,025
Phr16-Real	0,889	0,947	0,964	0,896	0,913	0,930	0,849	0,913	0,930	0,897	
Phr17-Norm	0,916	0,948	0,966	0,899	0,916	0,932	0,853	0,916	0,932	0,900	0,015
Phr17-Real	0,903	0,950	0,966	0,901	0,919	0,934	0,856	0,918	0,934	0,903	
Phr18-Norm	0,944	0,952	0,968	0,905	0,921	0,937	0,862	0,921	0,937	0,907	0,023
Phr18-Real	0,926	0,954	0,970	0,910	0,926	0,941	0,868	0,925	0,940	0,912	
Phr19-Norm	0,928	0,957	0,971	0,914	0,929	0,943	0,874	0,928	0,943	0,916	0,012
Phr19-Real	0,937	0,958	0,972	0,917	0,931	0,945	0,878	0,931	0,945	0,919	
Phr20-Norm	0,994	0,960	0,973	0,920	0,934	0,947	0,883	0,934	0,947	0,922	0,060
Phr20-Real	0,935	0,962	0,975	0,924	0,937	0,950	0,888	0,937	0,950	0,925	
Phr21-Norm	0,839	0,963	0,976	0,927	0,940	0,952	0,893	0,940	0,952	0,929	0,043
Phr21-Real	0,882	0,965	0,977	0,930	0,942	0,954	0,897	0,942	0,954	0,932	
Phr22-Norm	0,894	0,967	0,978	0,934	0,945	0,956	0,903	0,945	0,956	0,936	0,013
Phr22-Real	0,894	0,969	0,980	0,939	0,950	0,959	0,910	0,949	0,959	0,940	
Phr23-Norm	0,899	0,971	0,981	0,942	0,952	0,962	0,916	0,952	0,962	0,944	0,014
Phr23-Real	0,887	0,973	0,982	0,945	0,955	0,964	0,920	0,955	0,964	0,946	
Phr24-Norm	0,923	0,975	0,983	0,950	0,958	0,966	0,926	0,958	0,966	0,950	0,014
Phr24-Real	0,928	0,978	0,985	0,955	0,962	0,969	0,933	0,962	0,969	0,954	
Phr25-Norm	0,952	0,980	0,986	0,959	0,966	0,972	0,940	0,966	0,972	0,959	0,017
Phr25-Real	0,966	0,982	0,987	0,963	0,969	0,975	0,945	0,969	0,975	0,962	
Phr26-Norm	0,922	0,984	0,989	0,967	0,973	0,978	0,951	0,972	0,978	0,967	0,013
Phr26-Real	0,929	0,986	0,991	0,971	0,976	0,981	0,958	0,976	0,981	0,972	
Phr27-Norm	0,949	0,988	0,992	0,975	0,980	0,984	0,963	0,980	0,984	0,976	0,018
Phr27-Real	0,935	0,990	0,993	0,979	0,984	0,987	0,969	0,983	0,987	0,980	
Phr28-Norm	0,920	0,992	0,995	0,983	0,987	0,989	0,975	0,986	0,989	0,984	0,020
Phr28-Real	0,938	0,994	0,996	0,987	0,990	0,992	0,981	0,990	0,992	0,987	
Phr29-Norm	0,883	0,996	0,997	0,991	0,993	0,994	0,987	0,992	0,994	0,991	0,022
Phr29-Real	0,903	0,997	0,998	0,994	0,995	0,996	0,991	0,995	0,996	0,994	
Phr30-Norm	0,884	0,999	0,999	0,997	0,998	0,998	0,996	0,998	0,998	0,997	0,023
Phr30-Real	0,906	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	

Source: Elaborated by the author.

Table 40 – Oracle 01 – VozMe – Feature-based test oracles against normalization issues

	Ext 1	Ext 2	Ext 3	Ext 4	Ext 5	Ext 6	Ext 7	Ext 8	Ext 9	Ext 10	Ext 11	Ext 12	Ext 13	Ext 14	Ext 15	Ext 16	Ext 17	Enc. Dist.
Phr01-Norm	0.924	0.962	0.697	0.981	0.769	0.782	1.000	0.932	0.735	0.605	0.850	0.932	0.924	0.929	0.764	0.893	0.893	0.394
Phr01-Real	0.808	0.872	0.651	0.956	0.801	0.833	0.937	0.832	0.796	0.659	0.766	0.832	0.808	0.781	0.565	0.784	0.837	0.394
Phr02-Norm	0.747	0.801	0.763	0.993	0.827	0.845	0.958	0.753	0.760	0.648	0.873	0.753	0.747	0.754	0.491	0.611	0.875	0.402
Phr02-Real	0.886	0.930	0.684	0.957	0.779	0.815	0.944	0.906	0.799	0.592	0.873	0.906	0.886	0.874	0.566	0.737	0.845	0.402
Phr03-Norm	0.618	0.710	0.881	0.992	0.951	0.941	0.939	0.622	0.766	0.838	0.763	0.622	0.618	0.628	0.586	0.764	0.396	0.396
Phr03-Real	0.779	0.842	0.832	0.969	0.921	0.930	0.931	0.770	0.807	0.813	0.703	0.770	0.779	0.778	0.616	0.664	0.767	0.396
Phr04-Norm	0.772	0.873	0.703	0.993	0.770	0.783	1.000	0.804	0.737	0.541	0.826	0.804	0.772	0.758	0.611	0.747	0.861	0.387
Phr04-Real	0.912	0.976	0.672	0.961	0.776	0.806	0.981	0.936	0.820	0.593	0.767	0.609	0.912	0.901	0.729	0.870	0.867	0.387
Phr05-Norm	0.610	0.690	0.891	0.995	0.941	0.945	0.908	0.609	0.813	0.912	0.760	0.609	0.611	0.614	0.433	0.575	0.827	0.179
Phr05-Real	0.593	0.663	0.921	0.968	0.986	0.992	0.905	0.597	0.876	1.000	0.707	0.597	0.593	0.597	0.387	0.522	0.766	0.179
Phr06-Norm	0.683	0.732	1.000	0.986	1.000	1.000	0.944	0.665	0.882	0.978	0.720	0.665	0.683	0.716	0.548	0.599	0.804	0.259
Phr06-Real	0.618	0.682	0.880	0.961	0.985	0.988	0.933	0.618	0.929	0.869	0.728	0.618	0.618	0.618	0.460	0.543	0.764	0.259
Phr07-Norm	0.841	0.870	0.833	0.988	0.941	0.963	0.914	0.850	0.870	0.931	0.724	0.850	0.841	0.840	0.447	0.602	0.777	0.169
Phr07-Real	0.872	0.885	0.785	0.963	0.904	0.932	0.922	0.875	0.894	0.831	0.734	0.876	0.872	0.883	0.502	0.667	0.789	0.169
Phr08-Norm	0.710	0.791	0.914	0.991	0.963	0.979	0.929	0.736	0.813	0.892	0.709	0.736	0.710	0.706	0.455	0.619	0.738	0.213
Phr08-Real	0.776	0.829	0.806	0.958	0.936	0.961	0.912	0.788	0.883	0.826	0.718	0.788	0.776	0.782	0.456	0.603	0.749	0.213
Phr09-Norm	0.929	0.984	0.705	0.984	0.822	0.835	0.988	0.932	0.718	0.711	0.745	0.932	0.929	0.917	0.825	0.943	0.879	0.290
Phr09-Real	0.913	0.968	0.572	0.980	0.766	0.792	0.960	0.932	0.779	0.597	0.730	0.932	0.913	0.921	0.636	0.857	0.889	0.290
Phr10-Norm	0.793	0.853	0.759	0.981	0.889	0.900	0.940	0.809	0.767	0.754	0.733	0.809	0.793	0.802	0.633	0.717	0.792	0.086
Phr10-Real	0.805	0.858	0.809	0.986	0.894	0.909	0.940	0.811	0.794	0.756	0.734	0.811	0.804	0.803	0.576	0.718	0.769	0.086
Phr11-Norm	0.846	0.911	0.594	0.983	0.694	0.719	0.995	0.856	0.768	0.488	0.884	0.857	0.846	0.862	0.725	0.849	0.932	0.356
Phr11-Real	0.747	0.831	0.637	0.977	0.746	0.769	0.968	0.769	0.798	0.543	0.821	0.769	0.747	0.743	0.531	0.740	0.854	0.356
Phr12-Norm	0.617	0.697	0.783	0.984	0.845	0.847	0.980	0.617	0.803	0.633	0.861	0.617	0.617	0.624	0.449	0.584	0.832	0.248
Phr12-Real	0.540	0.626	0.815	0.984	0.878	0.880	0.975	0.539	0.838	0.685	0.833	0.539	0.540	0.552	0.357	0.482	0.817	0.248
Phr13-Norm	0.750	0.819	0.839	0.987	0.945	0.951	0.955	0.764	0.784	0.834	0.720	0.764	0.750	0.751	0.522	0.664	0.756	0.541
Phr13-Real	0.956	0.972	0.754	0.992	0.884	0.900	0.956	0.950	0.794	0.768	0.721	0.951	0.955	0.984	0.691	0.778	0.797	0.541
Phr14-Norm	0.722	0.782	0.864	0.992	0.926	0.928	0.945	0.718	0.844	0.816	0.732	0.718	0.722	0.747	0.644	0.723	0.791	0.111
Phr14-Real	0.688	0.762	0.881	0.994	0.909	0.908	0.944	0.684	0.851	0.799	0.740	0.684	0.688	0.714	0.593	0.762	0.816	0.111
Phr15-Norm	0.703	0.760	0.791	0.993	0.891	0.891	0.976	0.693	0.822	0.769	0.790	0.693	0.703	0.714	0.595	0.634	0.850	0.415
Phr15-Real	0.557	0.645	0.853	0.991	0.906	0.905	0.976	0.575	0.819	0.760	0.787	0.575	0.557	0.563	0.381	0.519	0.795	0.415
Phr16-Norm	0.561	0.678	0.820	1.000	0.878	0.878	0.969	0.581	0.776	0.720	0.804	0.581	0.561	0.552	0.466	0.588	0.859	0.266
Phr16-Real	0.668	0.777	0.818	0.997	0.858	0.865	0.950	0.674	0.782	0.718	0.778	0.675	0.668	0.670	0.528	0.623	0.874	0.266
Phr17-Norm	0.609	0.694	0.853	0.998	0.922	0.925	0.921	0.611	0.819	0.790	0.799	0.611	0.609	0.606	0.447	0.563	0.825	0.533
Phr17-Real	0.801	0.875	0.893	0.960	0.930	0.935	0.922	0.781	0.864	0.823	0.811	0.781	0.801	0.768	0.698	0.712	0.873	0.533
Phr18-Norm	0.795	0.838	0.700	0.989	0.836	0.853	0.986	0.813	0.782	0.661	0.809	0.813	0.795	0.805	0.500	0.614	0.812	0.200
Phr18-Real	0.850	0.889	0.667	0.954	0.810	0.840	0.961	0.851	0.822	0.618	0.848	0.851	0.849	0.850	0.558	0.729	0.854	0.200
Phr19-Norm	0.473	0.554	0.774	0.992	0.867	0.858	0.993	0.464	0.773	0.672	0.848	0.464	0.472	0.493	0.351	0.470	0.843	0.349
Phr19-Real	0.604	0.668	0.772	0.963	0.859	0.862	0.986	0.576	0.801	0.662	0.779	0.577	0.604	0.624	0.481	0.558	0.865	0.349
Phr20-Norm	0.562	0.645	0.890	0.995	0.915	0.920	0.963	0.561	0.825	0.801	0.781	0.561	0.563	0.570	0.359	0.499	0.821	0.123
Phr20-Real	0.600	0.663	0.816	0.973	0.902	0.909	0.955	0.585	0.824	0.771	0.769	0.586	0.600	0.608	0.398	0.522	0.835	0.123
Phr21-Norm	0.874	0.928	0.689	0.988	0.788	0.804	0.982	0.864	0.817	0.676	0.926	0.864	0.873	0.879	0.702	0.743	0.904	0.434
Phr21-Real	0.714	0.805	0.666	0.961	0.803	0.828	0.971	0.739	0.879	0.708	0.870	0.738	0.714	0.710	0.513	0.615	0.890	0.434
Phr22-Norm	0.733	0.812	0.785	0.988	0.859	0.871	0.967	0.753	0.738	0.705	0.870	0.753	0.733	0.731	0.555	0.658	0.808	0.207
Phr22-Real	0.776	0.831	0.816	0.956	0.919	0.936	0.925	0.790	0.825	0.773	0.683	0.790	0.776	0.778	0.511	0.632	0.757	0.207
Phr23-Norm	0.795	0.855	0.702	0.983	0.801	0.819	0.984	0.803	0.738	0.621	0.722	0.803	0.795	0.807	0.612	0.771	0.832	0.231
Phr23-Real	0.751	0.796	0.747	0.955	0.828	0.838	0.941	0.603	0.836	0.773	0.791	0.734	0.751	0.763	0.577	0.668	0.851	0.231
Phr24-Norm	0.687	0.750	0.851	0.985	0.947	0.953	0.945	0.705	0.784	0.830	0.734	0.705	0.686	0.693	0.455	0.570	0.756	0.502
Phr24-Real	0.495	0.596	0.882	0.957	0.973	0.973	0.938	0.511	0.861	0.858	0.712	0.511	0.495	0.498	0.323	0.452	0.741	0.502
Phr25-Norm	0.873	0.928	0.805	0.985	0.906	0.921	0.948	0.895	0.819	0.788	0.714	0.896	0.873	0.875	0.636	0.767	0.790	0.124
Phr25-Real	0.910	0.961	0.808	0.945	0.909	0.941	0.926	0.938	0.858	0.800	0.693	0.938	0.910	0.909	0.598	0.762	0.761	0.124
Phr26-Norm	0.967	0.986	0.742	0.975	0.832	0.860	0.966	0.987	0.813	0.672	0.813	0.987	0.967	0.962	0.635	0.750	0.826	0.130
Phr26-Real	1.000	1.000	0.739	0.953	0.853	0.891	0.935	1.000	0.873	0.717	0.759	1.000	1.000	1.000	0.612	0.761	0.801	0.130
Phr27-Norm	0.587	0.673	0.861	0.985	0.922	0.930	0.941	0.603	0.836	0.773	0.769	0.603	0.587	0.561	0.321	0.527	0.807	0.138
Phr27-Real	0.618	0.721	0.889	0.958	0.894	0.912	0.933	0.629	0.864	0.735	0.799	0.629	0.619	0.570	0.385	0.584	0.818	0.138
Phr28-Norm	0.711	0.797	0.794	0.985	0.909	0.916	0.945	0.728	0.826	0.781	0.752	0.728	0.711	0.708	0.541	0.649	0.787	0.171
Phr28-Real	0.721	0.802	0.894	0.948	0.943	0.959	0.932	0.732	0.909	0.842	0.732	0.721	0.721	0.728	0.577	0.685	0.756	0.171
Phr29-Norm	0.844	0.892	0.930	0.994	0.772	0.766	0.972	0.785	1.000	0.979	1.000	0.786	0.844	0.914	1.000	1.000	1.000	1.302
Phr29-Real	0.364	0.509	0.873	0.988	0.779	0.775	0.994	0.372	0.999	0.966	0.942	0.372	0.364	0.404	0.414	0.636	0.989	1.302
Phr30-Norm	0.779	0.949	0.736	0.962	0.862	0.885	0.939	0.800	0.877	0.756	0.745	0.800	0.767	0.767	0.467	0.577	0.820	0.366
Phr30-Real	0.841	0.878	0.788	0.983	0.888	0.903	0.947	0.830	0.875	0.766	0.745	0.830	0.841	0.849	0.670	0.777	0.818	0.366

Source: Elaborated by the author.

Table 41 – Oracle 02 – VozMe – Feature-based test oracles against normalization issues

	TDA 1	TDA 2	TDA 3	TDA 4	TDA 5	TDA 6	TDA 7	TDA 8	TDA 9	TDA 10	Euc. Dist.
Phr01-Norm	0,795	0,941	0,951	0,885	0,894	0,904	0,832	0,894	0,904	0,859	0,073
Phr01-Real	0,868	0,941	0,952	0,886	0,896	0,905	0,834	0,896	0,905	0,861	
Phr02-Norm	0,852	0,942	0,952	0,888	0,897	0,907	0,836	0,897	0,907	0,864	
Phr02-Real	0,878	0,943	0,953	0,890	0,899	0,909	0,839	0,900	0,909	0,866	0,026
Phr03-Norm	0,938	0,944	0,954	0,892	0,901	0,910	0,842	0,901	0,911	0,869	
Phr03-Real	0,944	0,946	0,955	0,893	0,903	0,912	0,844	0,903	0,912	0,871	0,008
Phr04-Norm	0,811	0,946	0,956	0,895	0,904	0,913	0,847	0,904	0,913	0,873	
Phr04-Real	0,858	0,947	0,957	0,898	0,906	0,915	0,850	0,907	0,915	0,876	0,048
Phr05-Norm	0,945	0,949	0,958	0,899	0,908	0,917	0,853	0,908	0,917	0,878	
Phr05-Real	0,978	0,949	0,958	0,901	0,909	0,918	0,855	0,910	0,918	0,880	0,033
Phr06-Norm	0,966	0,950	0,959	0,902	0,911	0,919	0,857	0,912	0,919	0,881	
Phr06-Real	0,963	0,951	0,960	0,904	0,912	0,920	0,859	0,912	0,920	0,883	0,005
Phr07-Norm	0,986	0,952	0,960	0,906	0,914	0,922	0,862	0,914	0,922	0,885	
Phr07-Real	0,970	0,953	0,961	0,908	0,916	0,924	0,865	0,916	0,924	0,888	0,016
Phr08-Norm	1,000	0,954	0,963	0,910	0,918	0,926	0,868	0,919	0,926	0,891	
Phr08-Real	0,988	0,956	0,963	0,913	0,920	0,928	0,872	0,921	0,928	0,894	0,014
Phr09-Norm	0,874	0,957	0,964	0,915	0,922	0,930	0,875	0,922	0,930	0,896	
Phr09-Real	0,833	0,957	0,965	0,916	0,923	0,931	0,877	0,924	0,931	0,898	0,041
Phr10-Norm	0,927	0,958	0,965	0,918	0,925	0,933	0,879	0,926	0,932	0,900	
Phr10-Real	0,926	0,959	0,966	0,920	0,927	0,934	0,882	0,927	0,934	0,903	0,006
Phr11-Norm	0,754	0,960	0,967	0,922	0,929	0,936	0,885	0,930	0,936	0,906	
Phr11-Real	0,812	0,961	0,968	0,923	0,930	0,937	0,887	0,931	0,938	0,908	0,058
Phr12-Norm	0,840	0,962	0,969	0,925	0,932	0,939	0,890	0,933	0,939	0,910	
Phr12-Real	0,859	0,964	0,970	0,928	0,935	0,941	0,894	0,935	0,941	0,913	0,020
Phr13-Norm	0,955	0,964	0,971	0,931	0,937	0,943	0,898	0,937	0,943	0,916	
Phr13-Real	0,917	0,966	0,972	0,933	0,939	0,945	0,901	0,940	0,945	0,919	0,039
Phr14-Norm	0,922	0,967	0,973	0,934	0,940	0,946	0,904	0,941	0,947	0,921	
Phr14-Real	0,902	0,967	0,974	0,936	0,942	0,947	0,906	0,942	0,948	0,923	0,020
Phr15-Norm	0,876	0,969	0,974	0,938	0,943	0,949	0,908	0,944	0,949	0,924	
Phr15-Real	0,892	0,970	0,975	0,940	0,944	0,950	0,910	0,945	0,950	0,926	0,017
Phr16-Norm	0,864	0,970	0,976	0,941	0,946	0,951	0,913	0,947	0,951	0,928	
Phr16-Real	0,858	0,971	0,976	0,943	0,948	0,953	0,916	0,948	0,953	0,930	0,008
Phr17-Norm	0,906	0,972	0,977	0,944	0,949	0,954	0,918	0,950	0,954	0,932	
Phr17-Real	0,919	0,972	0,978	0,946	0,951	0,955	0,920	0,951	0,955	0,933	0,014
Phr18-Norm	0,881	0,974	0,979	0,948	0,953	0,957	0,923	0,953	0,957	0,936	
Phr18-Real	0,879	0,975	0,980	0,951	0,955	0,959	0,927	0,955	0,960	0,940	0,008
Phr19-Norm	0,835	0,977	0,981	0,953	0,958	0,961	0,931	0,958	0,961	0,943	
Phr19-Real	0,848	0,978	0,982	0,955	0,959	0,963	0,934	0,959	0,963	0,944	0,014
Phr20-Norm	0,902	0,978	0,982	0,957	0,960	0,964	0,936	0,961	0,964	0,947	
Phr20-Real	0,894	0,980	0,983	0,959	0,962	0,966	0,940	0,962	0,966	0,949	0,010
Phr21-Norm	0,815	0,980	0,984	0,961	0,964	0,967	0,942	0,965	0,967	0,951	
Phr21-Real	0,862	0,981	0,985	0,962	0,965	0,969	0,944	0,966	0,968	0,952	0,047
Phr22-Norm	0,881	0,983	0,986	0,965	0,967	0,970	0,947	0,968	0,970	0,956	
Phr22-Real	0,954	0,984	0,987	0,968	0,970	0,972	0,952	0,970	0,973	0,959	0,073
Phr23-Norm	0,840	0,986	0,988	0,970	0,972	0,975	0,956	0,973	0,975	0,962	
Phr23-Real	0,855	0,986	0,988	0,972	0,974	0,976	0,958	0,974	0,976	0,964	0,016
Phr24-Norm	0,950	0,988	0,989	0,975	0,976	0,979	0,962	0,977	0,978	0,968	
Phr24-Real	0,959	0,989	0,990	0,978	0,979	0,981	0,967	0,980	0,980	0,971	0,013
Phr25-Norm	0,954	0,991	0,991	0,981	0,981	0,982	0,971	0,982	0,983	0,974	
Phr25-Real	0,997	0,991	0,992	0,983	0,983	0,984	0,974	0,984	0,985	0,977	0,043
Phr26-Norm	0,886	0,993	0,993	0,985	0,986	0,987	0,978	0,987	0,987	0,980	
Phr26-Real	0,930	0,994	0,995	0,987	0,988	0,990	0,981	0,989	0,990	0,984	0,045
Phr27-Norm	0,945	0,995	0,996	0,990	0,990	0,992	0,985	0,991	0,991	0,987	
Phr27-Real	0,926	0,996	0,997	0,992	0,992	0,994	0,988	0,993	0,993	0,990	0,019
Phr28-Norm	0,922	0,997	0,998	0,994	0,994	0,995	0,991	0,995	0,995	0,992	
Phr28-Real	0,977	0,998	0,998	0,996	0,996	0,996	0,994	0,997	0,996	0,995	0,055
Phr29-Norm	0,718	0,999	0,999	0,997	0,997	0,997	0,995	0,998	0,997	0,996	
Phr29-Real	0,736	0,999	0,999	0,997	0,997	0,997	0,996	0,998	0,997	0,996	0,017
Phr30-Norm	0,921	0,999	0,999	0,998	0,998	0,998	0,998	0,998	0,998	0,998	
Phr30-Real	0,913	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,010

Source: Elaborated by the author.

RESULTS OF FEATURE-BASED TEST ORACLES EVALUATING TTS WITH REDUCED AUDIO DATABASES (SEC. 5.4)

Find bellow all of the tables regarding some oracle configuration.

Table 42 – Oracle 01 – Feature-based test oracles evaluating TTT with reduced audio database

Feature Extractors' Results																		th 1	th 2	th 3	
	E01	E02	E03	E04	E05	E06	E07	E08	E09	E10	E11	E12	E13	E14	E15	E16	E17	Sent.	Dist	pass	fail
01Max	0.921	0.917	0.719	0.993	0.747	0.744	0.964	0.923	0.709	0.580	0.866	0.923	0.921	0.937	0.965	0.922	0.930	0.000	0.000	pass	pass
01Md	0.775	0.837	0.847	0.996	0.766	0.759	0.973	0.794	0.710	0.589	0.818	0.795	0.774	0.716	0.782	0.929	0.885	0.431	0.431	fail	fail
01Min	0.932	0.939	0.635	0.983	0.753	0.748	1.000	0.911	0.648	0.532	0.874	0.911	0.932	0.932	0.958	0.923	0.969	0.132	0.132	pass	fail
01MMin	0.783	0.852	0.621	0.969	0.774	0.769	0.938	0.790	0.732	0.558	0.844	0.790	0.782	0.745	0.821	0.931	0.953	0.387	0.387	fail	fail
stdev	0.086	0.049	0.104	0.012	0.013	0.011	0.026	0.072	0.036	0.026	0.025	0.072	0.086	0.118	0.094	0.005	0.037				
10Max	0.766	0.820	0.772	0.988	0.834	0.820	0.964	0.782	0.825	0.604	0.896	0.782	0.765	0.730	0.759	0.809	0.926	0.000	0.000	pass	pass
10Md	0.772	0.826	0.830	0.997	0.816	0.816	0.945	0.785	0.934	0.614	0.868	0.785	0.771	0.753	0.712	0.817	0.870	0.229	0.229	pass	fail
10Min	0.758	0.813	0.747	0.981	0.859	0.859	0.903	0.776	0.942	0.721	0.838	0.776	0.758	0.679	0.702	0.763	0.908	0.151	0.151	pass	fail
10MMin	0.786	0.854	0.665	0.992	0.856	0.857	0.888	0.806	0.857	0.679	0.832	0.806	0.786	0.730	0.820	0.933	0.917	0.214	0.214	pass	fail
stdev	0.012	0.018	0.068	0.007	0.020	0.023	0.036	0.013	0.057	0.055	0.029	0.013	0.012	0.031	0.054	0.073	0.025				
11Max	0.631	0.704	0.823	0.997	0.802	0.790	0.934	0.673	0.757	0.647	0.843	0.673	0.631	0.537	0.551	0.704	0.899	0.000	0.000	pass	pass
11Md	0.569	0.612	0.852	0.979	0.859	0.865	0.906	0.609	0.588	0.762	0.817	0.609	0.569	0.503	0.436	0.600	0.913	0.320	0.320	pass	fail
11Min	0.531	0.563	0.711	0.989	0.843	0.833	0.928	0.586	0.771	0.675	0.831	0.586	0.531	0.489	0.373	0.606	0.961	0.345	0.345	pass	fail
11MMin	0.706	0.759	0.586	0.978	0.722	0.722	0.971	0.721	0.655	0.519	0.827	0.721	0.706	0.626	0.591	0.784	1.000	0.375	0.375	fail	fail
stdev	0.077	0.088	0.121	0.009	0.061	0.062	0.027	0.061	0.087	0.101	0.011	0.061	0.077	0.062	0.101	0.088	0.046				
20Max	0.655	0.731	0.874	0.989	0.876	0.859	0.899	0.704	0.910	0.645	0.873	0.703	0.655	0.603	0.689	0.660	0.891	0.000	0.000	pass	pass
20Md	0.657	0.729	0.978	0.989	0.919	0.907	0.891	0.693	1.000	0.703	0.906	0.693	0.656	0.612	0.633	0.688	0.889	0.178	0.178	pass	fail
20Min	0.747	0.821	0.641	0.991	0.865	0.858	0.871	0.775	0.710	0.648	0.855	0.775	0.746	0.681	0.761	0.843	0.958	0.425	0.425	fail	fail
20MMin	0.696	0.776	0.647	0.985	0.854	0.848	0.869	0.726	0.839	0.645	0.882	0.726	0.696	0.642	0.711	0.764	0.948	0.284	0.284	fail	fail
stdev	0.043	0.044	0.168	0.003	0.028	0.027	0.015	0.037	0.123	0.029	0.021	0.036	0.043	0.035	0.053	0.082	0.037				
21Max	0.694	0.746	0.812	0.992	0.842	0.852	0.953	0.708	0.945	0.688	0.949	0.708	0.694	0.625	0.630	0.810	0.924	0.000	0.000	pass	pass
21Md	0.614	0.686	0.912	0.988	0.839	0.845	0.914	0.650	0.892	0.708	0.867	0.650	0.614	0.545	0.568	0.718	0.915	0.252	0.252	fail	fail
21Min	0.684	0.738	0.732	0.973	0.799	0.819	0.936	0.713	0.923	0.681	0.914	0.713	0.684	0.626	0.610	0.753	0.960	0.131	0.131	pass	fail
21MMin	0.708	0.754	0.707	0.992	0.811	0.822	0.968	0.735	0.962	0.656	1.000	0.734	0.708	0.679	0.659	0.772	0.978	0.165	0.165	pass	fail
stdev	0.042	0.031	0.092	0.009	0.021	0.016	0.023	0.036	0.030	0.022	0.056	0.036	0.042	0.055	0.038	0.038	0.030				
30Max	0.723	0.750	0.845	0.971	0.847	0.836	0.929	0.731	0.992	0.591	0.871	0.731	0.722	0.697	0.656	0.693	0.894	0.000	0.000	pass	pass
30Md	0.732	0.743	0.824	0.974	0.870	0.861	0.904	0.736	0.954	0.649	0.848	0.735	0.732	0.732	0.629	0.605	0.875	0.134	0.134	pass	fail
30Min	0.759	0.757	0.676	0.995	0.851	0.852	0.888	0.753	0.833	0.616	0.842	0.753	0.759	0.770	0.700	0.728	0.921	0.266	0.266	pass	fail
30MMin	0.683	0.701	0.617	0.971	0.853	0.845	0.896	0.699	0.807	0.614	0.843	0.699	0.683	0.673	0.589	0.609	0.943	0.333	0.333	pass	fail
stdev	0.031	0.025	0.111	0.011	0.010	0.011	0.018	0.023	0.090	0.024	0.014	0.023	0.031	0.042	0.046	0.062	0.030				

Source: Elaborated by the author.

Table 43 – Oracle 02 – Feature-based test oracles evaluating TTT with reduced audio database

Results of Feature Extractors' Dimensions														
	TAMM 1	TAMM 2	TAMM 3	TAMM 4	TAMM 5	TAMM 6	TAMM 7	TAMM 8	TAMM 9	TAMM 10	Dist	th 1 0,134	th 2 0,067	th 3 0,033
01Max	0,761	0,003	0,004	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	pass	pass	pass
01Md	0,777	0,010	0,011	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,019	pass	pass	pass
01Min	0,773	0,023	0,025	0,001	0,001	0,001	0,000	0,001	0,001	0,000	0,031	pass	pass	pass
01MMin	0,796	0,017	0,018	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,040	pass	pass	fail
stdev	0,015	0,008	0,009	0,000	0,000	0,000	0,000	0,000	0,000	0,000				
10Max	0,842	0,297	0,309	0,088	0,092	0,095	0,026	0,092	0,095	0,029	0,000	pass	pass	pass
10Md	0,857	0,307	0,319	0,094	0,098	0,102	0,029	0,098	0,102	0,032	0,026	pass	pass	pass
10Min	0,899	0,325	0,339	0,105	0,110	0,115	0,034	0,110	0,115	0,039	0,083	pass	fail	fail
10MMin	0,897	0,316	0,329	0,100	0,104	0,108	0,032	0,104	0,109	0,036	0,068	pass	fail	fail
stdev	0,029	0,012	0,013	0,007	0,008	0,008	0,003	0,008	0,008	0,004				
11Max	0,851	0,332	0,348	0,110	0,115	0,121	0,037	0,116	0,121	0,042	0,000	pass	pass	pass
11Md	0,894	0,338	0,354	0,115	0,120	0,125	0,039	0,120	0,125	0,044	0,046	pass	pass	fail
11Min	0,868	0,350	0,365	0,122	0,128	0,133	0,043	0,128	0,134	0,049	0,042	pass	pass	fail
11MMin	0,755	0,344	0,360	0,119	0,124	0,129	0,041	0,124	0,130	0,047	0,099	pass	fail	fail
stdev	0,061	0,008	0,008	0,005	0,005	0,005	0,003	0,005	0,005	0,003				
20Max	0,882	0,615	0,620	0,378	0,381	0,384	0,233	0,381	0,384	0,238	0,000	pass	pass	pass
20Md	0,913	0,625	0,630	0,391	0,394	0,397	0,245	0,394	0,397	0,250	0,048	pass	pass	fail
20Min	0,882	0,644	0,649	0,414	0,417	0,421	0,267	0,417	0,420	0,273	0,102	pass	fail	fail
20MMin	0,862	0,635	0,640	0,404	0,406	0,409	0,256	0,406	0,409	0,262	0,074	pass	fail	fail
stdev	0,021	0,012	0,012	0,016	0,016	0,016	0,015	0,016	0,016	0,015				
21Max	0,895	0,651	0,656	0,424	0,427	0,431	0,276	0,427	0,430	0,282	0,000	pass	pass	pass
21Md	0,898	0,658	0,663	0,434	0,437	0,440	0,285	0,437	0,440	0,292	0,027	pass	pass	pass
21Min	0,857	0,671	0,677	0,450	0,454	0,458	0,302	0,454	0,458	0,310	0,085	pass	fail	fail
21MMin	0,873	0,665	0,670	0,442	0,445	0,449	0,294	0,445	0,449	0,301	0,056	pass	pass	fail
stdev	0,019	0,008	0,009	0,011	0,012	0,012	0,011	0,012	0,012	0,012				
30Max	0,845	0,978	0,978	0,958	0,957	0,957	0,937	0,957	0,957	0,936	0,000	pass	pass	pass
30Md	0,877	0,986	0,986	0,972	0,972	0,972	0,959	0,972	0,972	0,959	0,057	pass	pass	fail
30Min	0,875	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,138	fail	fail	fail
30MMin	0,864	0,993	0,993	0,987	0,986	0,986	0,980	0,986	0,986	0,979	0,093	pass	fail	fail
stdev	0,015	0,009	0,009	0,018	0,018	0,018	0,027	0,019	0,019	0,027				

Source: Elaborated by the author.

Table 44 – Oracle 03 – Feature-based test oracles evaluating TTT with reduced audio database

Sent.	Results of Chroma's Dimensions													th 1 0,169	th 2 0,084	th 3 0,042
	Ch 01	Ch 02	Ch 03	Ch 04	Ch 05	Ch 06	Ch 07	Ch 08	Ch 09	Ch 10	Ch 11	Ch 12	Dist			
01Max	0,742	0,742	0,740	0,737	0,738	0,739	0,740	0,740	0,738	0,738	0,738	0,738	0,000	pass	pass	pass
01Md	0,767	0,764	0,761	0,758	0,759	0,761	0,763	0,764	0,762	0,762	0,764	0,763	0,081	pass	fail	fail
01Min	0,761	0,761	0,762	0,760	0,760	0,760	0,760	0,761	0,760	0,761	0,762	0,760	0,074	pass	pass	fail
01MMin	0,760	0,761	0,761	0,758	0,757	0,759	0,759	0,759	0,758	0,761	0,763	0,763	0,072	pass	fail	fail
stdev	0,010	0,010	0,011	0,011	0,010	0,010	0,010	0,011	0,011	0,012	0,012	0,012				
10Max	0,827	0,826	0,823	0,820	0,819	0,821	0,823	0,824	0,824	0,826	0,827	0,827	0,000	pass	pass	pass
10Md	0,817	0,815	0,812	0,809	0,808	0,806	0,807	0,809	0,808	0,810	0,809	0,810	0,049	pass	fail	fail
10Min	0,848	0,848	0,849	0,848	0,846	0,848	0,847	0,846	0,843	0,845	0,846	0,845	0,079	pass	pass	fail
10MMin	0,864	0,865	0,865	0,864	0,863	0,865	0,864	0,864	0,862	0,863	0,863	0,864	0,138	pass	fail	fail
stdev	0,021	0,022	0,024	0,025	0,025	0,026	0,025	0,024	0,023	0,023	0,023	0,023				
11Max	0,795	0,793	0,790	0,787	0,786	0,788	0,789	0,792	0,793	0,794	0,795	0,795	0,000	pass	pass	pass
11Md	0,876	0,877	0,876	0,871	0,869	0,870	0,871	0,873	0,873	0,877	0,879	0,879	0,286	fail	fail	fail
11Min	0,826	0,830	0,831	0,832	0,831	0,835	0,835	0,837	0,835	0,839	0,839	0,838	0,148	pass	pass	fail
11MMin	0,735	0,737	0,737	0,735	0,734	0,733	0,733	0,735	0,735	0,739	0,741	0,741	0,192	fail	fail	fail
stdev	0,059	0,059	0,059	0,059	0,058	0,059	0,060	0,060	0,059	0,059	0,059	0,059				
20Max	0,893	0,892	0,889	0,884	0,883	0,882	0,884	0,887	0,887	0,889	0,890	0,890	0,000	pass	pass	pass
20Md	0,937	0,937	0,933	0,928	0,926	0,922	0,924	0,925	0,925	0,926	0,926	0,926	0,140	pass	fail	fail
20Min	0,884	0,886	0,887	0,885	0,886	0,886	0,887	0,890	0,889	0,888	0,888	0,887	0,013	pass	pass	pass
20MMin	0,852	0,854	0,855	0,855	0,856	0,858	0,859	0,860	0,859	0,861	0,860	0,858	0,107	pass	fail	fail
stdev	0,035	0,034	0,032	0,030	0,029	0,026	0,027	0,027	0,027	0,027	0,027	0,028				
21Max	0,864	0,863	0,862	0,861	0,858	0,859	0,860	0,860	0,859	0,862	0,864	0,865	0,000	pass	pass	pass
21Md	0,840	0,839	0,839	0,839	0,841	0,842	0,842	0,845	0,843	0,843	0,842	0,842	0,070	pass	fail	fail
21Min	0,809	0,807	0,804	0,802	0,801	0,800	0,802	0,803	0,803	0,803	0,802	0,803	0,202	fail	pass	fail
21MMin	0,799	0,800	0,802	0,803	0,804	0,805	0,803	0,802	0,800	0,803	0,805	0,806	0,204	fail	fail	fail
stdev	0,030	0,029	0,029	0,029	0,028	0,029	0,029	0,029	0,030	0,029	0,030	0,030				
30Max	0,848	0,848	0,846	0,844	0,844	0,847	0,849	0,852	0,852	0,855	0,855	0,854	0,000	pass	pass	pass
30Md	0,857	0,857	0,856	0,855	0,854	0,856	0,856	0,858	0,858	0,861	0,861	0,861	0,028	pass	fail	pass
30Min	0,857	0,858	0,856	0,857	0,859	0,863	0,864	0,867	0,866	0,869	0,871	0,869	0,048	pass	pass	fail
30MMin	0,842	0,844	0,846	0,848	0,849	0,856	0,856	0,859	0,858	0,862	0,862	0,862	0,022	pass	fail	pass
stdev	0,007	0,007	0,006	0,006	0,006	0,007	0,006	0,006	0,006	0,006	0,006	0,006				

Source: Elaborated by the author.

Table 45 – Oracle 04 – Feature-based test oracles evaluating TTT with reduced audio database

Results of LPC's dimensions																
	LPC 1	LPC 2	LPC 3	LPC 4	LPC 5	LPC 6	LPC 7	LPC 8	LPC 9	Dist	th 1	th 2	th 3			
01Max	0.815	0.693	0.361	0.648	0.578	0.572	0.605	0.627	0.635	0.000	pass	pass	pass			
01Md	0.871	0.693	0.511	0.692	0.597	0.606	0.627	0.677	0.516	0.216	pass	pass	fail			
01Min	0.808	0.634	0.633	0.808	0.695	0.326	0.587	0.589	0.289	0.547	pass	fail	fail			
01MMin	0.870	0.680	0.755	0.796	0.805	0.424	0.550	0.497	0.511	0.538	pass	fail	fail			
stdev	0.034	0.028	0.169	0.078	0.104	0.131	0.033	0.076	0.144							
10Max	0.872	0.583	0.348	0.702	0.511	0.637	0.703	0.743	0.793	0.000	pass	pass	pass			
10Md	0.881	0.684	0.311	0.766	0.228	0.546	0.807	0.754	0.905	0.357	pass	fail	fail			
10Min	0.878	0.694	0.369	0.584	0.508	0.583	0.763	0.775	0.706	0.205	pass	pass	fail			
10MMin	0.869	0.647	0.348	0.606	0.660	0.447	0.716	0.717	0.849	0.275	pass	pass	fail			
stdev	0.005	0.050	0.024	0.085	0.180	0.080	0.047	0.024	0.085							
11Max	0.929	0.705	0.462	0.471	0.300	0.617	0.758	0.766	0.968	0.000	pass	pass	pass			
11Md	0.954	0.660	0.598	0.735	0.491	0.552	0.759	0.870	1.000	0.379	pass	fail	fail			
11Min	0.960	0.644	0.886	0.546	0.734	0.652	0.640	0.764	0.934	0.628	pass	fail	fail			
11MMin	0.900	0.670	0.622	0.488	0.589	0.491	0.734	0.680	0.506	0.591	pass	fail	fail			
stdev	0.027	0.026	0.177	0.121	0.182	0.071	0.056	0.078	0.232							
20Max	0.907	0.641	0.186	0.592	0.155	0.670	0.663	0.821	0.603	0.000	pass	pass	pass			
20Md	0.910	0.623	0.367	0.546	0.262	0.638	0.787	0.697	0.681	0.290	pass	pass	fail			
20Min	0.877	0.634	0.344	0.619	0.242	0.427	0.654	0.756	0.718	0.333	pass	fail	fail			
20MMin	0.897	0.697	0.399	0.718	0.336	0.518	0.761	0.762	0.644	0.368	pass	fail	fail			
stdev	0.015	0.033	0.095	0.073	0.074	0.112	0.068	0.051	0.049							
21Max	0.912	0.763	0.414	0.501	0.235	0.493	0.731	0.800	0.632	0.000	pass	pass	pass			
21Md	0.936	0.770	0.483	0.758	0.239	0.367	0.655	0.668	0.658	0.334	pass	fail	fail			
21Min	0.911	0.738	0.516	0.595	0.297	0.303	0.601	0.759	0.691	0.286	pass	pass	fail			
21MMin	0.896	0.710	0.457	0.385	0.288	0.366	0.691	0.766	0.770	0.244	pass	pass	fail			
stdev	0.017	0.027	0.043	0.158	0.032	0.080	0.055	0.056	0.060							
30Max	0.889	0.771	0.651	0.703	0.535	0.505	0.592	0.560	0.702	0.000	pass	pass	pass			
30Md	0.888	0.801	0.689	0.741	0.388	0.520	0.617	0.544	0.802	0.192	pass	pass	fail			
30Min	0.877	0.802	0.648	0.653	0.594	0.543	0.618	0.570	0.693	0.097	pass	pass	pass			
30MMin	0.901	0.736	0.632	0.742	0.532	0.453	0.710	0.652	0.633	0.182	pass	pass	fail			
stdev	0.010	0.031	0.024	0.042	0.088	0.038	0.052	0.048	0.070							

Source: Elaborated by the author.

Table 46 – Oracle 05 – Feature-based test oracles evaluating TTT with reduced audio database

Results of MFCC's dimensions															
	MFCC 1	MFCC 2	MFCC 3	MFCC 4	MFCC 5	MFCC 6	MFCC 7	MFCC 8	MFCC 9	MFCC 10	MFCC 11	MFCC 12	MFCC 13	Dist	th 1 0.650
01Max	0.967	0.785	0.662	0.598	0.852	0.656	0.344	0.241	0.826	0.553	0.824	0.856	0.914	0.000	pass
01Md	0.954	0.799	0.671	0.477	0.905	0.688	0.017	0.110	1.000	0.363	0.941	0.934	0.901	0.479	pass
01MMin	0.947	0.807	0.427	0.438	0.746	0.554	0.442	0.514	0.623	0.684	0.808	0.823	0.893	0.497	fail
01Min	0.960	0.799	0.521	0.726	0.756	0.451	0.331	0.442	0.751	0.599	0.738	0.798	0.923	0.383	fail
stdev	0.009	0.009	0.118	0.130	0.077	0.107	0.185	0.185	0.158	0.136	0.084	0.059	0.013	0.185	pass
10Max	0.949	0.807	0.781	0.503	0.925	0.585	0.089	0.071	0.764	0.857	0.774	0.864	0.880	0.000	pass
10Md	0.927	0.796	0.689	0.729	0.998	0.578	0.207	0.031	0.769	0.803	0.798	0.938	0.892	0.301	pass
10MMin	0.902	0.812	0.507	0.927	0.972	0.468	0.057	0.407	0.808	0.638	0.765	0.958	0.938	0.670	fail
10Min	0.917	0.821	0.587	0.748	0.962	0.575	0.208	0.393	0.784	0.680	0.775	0.957	0.949	0.513	pass
stdev	0.020	0.011	0.119	0.174	0.030	0.056	0.079	0.202	0.020	0.103	0.014	0.044	0.034	0.202	pass
11Max	0.949	0.835	0.604	0.177	0.637	0.459	0.257	0.324	0.701	0.634	0.793	0.800	0.916	0.000	pass
11Md	0.943	0.876	0.631	0.237	0.606	0.372	0.058	0.219	0.751	0.848	0.686	0.810	0.921	0.353	fail
11MMin	0.974	0.834	0.462	0.392	0.505	0.212	0.375	0.585	0.678	0.685	0.655	0.723	0.960	0.508	fail
11Min	0.948	0.884	0.500	0.058	0.499	0.293	0.514	0.640	0.781	0.477	0.766	0.832	0.982	0.527	fail
stdev	0.014	0.026	0.081	0.139	0.070	0.106	0.193	0.203	0.047	0.153	0.065	0.047	0.032	0.203	pass
20Max	0.938	0.848	0.934	0.146	0.703	0.486	0.434	0.511	0.477	1.000	0.628	0.747	0.864	0.000	pass
20Md	0.932	0.868	0.837	0.057	0.739	0.553	0.559	0.514	0.559	0.892	0.641	0.766	0.875	0.242	pass
20MMin	0.933	0.855	0.620	0.186	0.711	0.536	0.818	0.860	0.569	0.696	0.731	0.862	0.904	0.706	fail
20Min	0.938	0.856	0.660	0.261	0.703	0.395	0.462	0.603	0.628	0.773	0.645	0.795	0.931	0.433	fail
stdev	0.003	0.008	0.148	0.085	0.017	0.071	0.175	0.164	0.062	0.134	0.047	0.050	0.030	0.175	pass
21Max	0.946	0.807	0.641	0.150	0.609	0.361	0.125	0.241	0.772	0.475	0.842	0.851	0.983	0.000	pass
21Md	0.943	0.815	0.500	0.247	0.567	0.401	0.079	0.130	0.829	0.519	0.852	0.782	0.955	0.241	pass
21MMin	0.952	0.783	0.450	0.476	0.588	0.372	0.183	0.346	0.585	0.763	0.815	0.814	1.000	0.528	fail
21Min	0.954	0.817	0.444	0.584	0.521	0.257	0.195	0.353	0.702	0.487	0.781	0.745	0.963	0.533	fail
stdev	0.005	0.015	0.092	0.200	0.037	0.063	0.053	0.105	0.105	0.136	0.032	0.046	0.020	0.200	pass
30Max	0.937	0.869	0.427	0.263	0.648	0.407	0.047	0.148	0.781	0.595	0.867	0.822	0.862	0.000	pass
30Md	0.932	0.875	0.369	0.358	0.674	0.426	0.162	0.352	0.709	0.650	0.811	0.801	0.859	0.283	pass
30MMin	0.932	0.876	0.326	0.486	0.621	0.381	0.448	0.603	0.549	0.656	0.773	0.780	0.914	0.708	fail
30Min	0.924	0.837	0.303	0.477	0.623	0.365	0.459	0.697	0.578	0.684	0.761	0.835	0.954	0.778	fail
stdev	0.005	0.018	0.055	0.106	0.025	0.027	0.207	0.249	0.110	0.037	0.048	0.024	0.045	0.249	pass

Source: Elaborated by the author.