

REDUCING THE DISTANCE BETWEEN REQUIREMENTS ENGINEERING AND VERIFICATION

Waleed Abdeen

Blekinge Institute of Technology
Licentiate Dissertation Series No. 2022:04
Department of Software Engineering



Reducing the Distance Between Requirements Engineering and Verification

Waleed Abdeen

Blekinge Institute of Technology Licentiate Dissertation Series
No 2022:04

Reducing the Distance Between Requirements Engineering and Verification

Waleed Abdeen

Licentiate Dissertation in
Software Engineering



Department of Software Engineering
Blekinge Institute of Technology
SWEDEN

2022 Waleed Abdeen
Department of Software Engineering
Publisher: Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Printed by Exakta Group, Sweden, 2022
ISBN: 978-91-7295-442-7
ISSN: 1650-2140
urn:nbn:se:bth-23570

To my parents.

Abstract

Background Requirements engineering and verification (REV) processes play essential roles in software product development. There are physical and non-physical distances between entities (actors, artifacts, and activities) in these processes. Current practices that reduce the distances, such as automated testing and alignment of document structure and tracing only partially close the above mentioned gap.

Objective The aim of this thesis is to investigate solutions w.r.t their ability to reduce the distances between requirements engineering and verification. Two techniques that are explored in this thesis are automated testing (model-based testing, MBT) and alignment of document structure and tracing (traceability).

Method The research methods used in this thesis are systematic mapping study, software requirements mining, case study, literature survey, validation study, and design science.

Results MBT and traceability are effective in reducing the distance between requirements and verification. However, both activities have some shortcoming that needs to be addressed when used for that purpose. Current MBT techniques in the context of software performance do not attain all the goals of MBT: 1) requirements validation, 2) checking the testability of requirements, and 3) the generation of an efficient test suite. These goals are essential to reduce the distance. We developed and assessed performance requirements verification and test environment generation approach to tackle these shortcomings. Also, traceability between requirements and verification suffers from the low granularity of trace links and does not support the verification of all requirements. We propose the use of taxonomic trace links to trace and align the structure of requirements specifications and verification artifacts. The results from the validation study show that the solution is feasible in practice. However, this comes with challenges that need to be addressed.

Conclusion MBT and improved traceability reduce multiple distances between actors, artifacts, and activities in the requirements engineering and verification process. MBT is most effective in reducing the distances when the model used is built from the requirements. Traceability is essential in easing access to relevant information when needed and should not be seen as an overhead. When creating trace links, we need to consider the difference in the abstraction, structure, and time between the linked artifacts

Acknowledgment

I like to start by thanking my supervisors, Dr. Michael Unterkalmsteiner and Dr. Krzysztof Wnuk, for their continuous support throughout this thesis. Michael has never hesitated to dedicate the time for our discussion whenever I needed. He always provided me with constructive feedback and was patient with me even when I had stupid questions. I learned from him more than just research. Also, I like to thank Krzysztof for all the support, guidance, and help and for being there to talk whenever things get tough.

Thanks to my colleagues at SERL Sweden and DIPT, who provided a good working environment and made me always feel welcome. I had a lot of fun with my fellows inside and outside BTH. They made the journey more fun. Thanks to Umar Iftikhar for the fruitful discussions and to Andreas Bauer for being a colleague and a friend.

The work done in this thesis was funded by the D-CAT and SERT projects. I thank everyone from the companies Trafikverket and HOCHTIEF Vicon, who was involved in this reasetch.

Thanks to my parents for raising me, my father, who spent his life giving the best to our family, and my mother, the first teacher I had and the most loving and caring person I know. My sisters Lama and Manal and my brother Mohammad, thanks for looking after me.

Last but not least, I would like to thank my fiance Alaa for all her love and support, for all the long nights, and for giving me the motivation to finish this thesis. Looking forward to our life together.

Preface

Included Papers

Chapter 2 Waleed Abdeen, Xingru Chen, and Michael Unterkalmsteiner. “An Approach for Performance Requirements Verification and Test Environments Generation.” *Requirements Engineering*, April 13, 2022. <https://doi.org/10.1007/s00766-022-00379-3>.

Chapter 3 Waleed Abdeen, Krzysztof Wnuk, Michael Unterkalmsteiner, and Alexandros Chirtoglou. “Challenges of Requirements Communication and Digital Assets Verification in Infrastructure Projects.” submitted to *Information and Software Technology Journal*, April 5, 2022.

Chapter 4 Waleed Abdeen, Michael Unterkalmsteiner, Alexandros Chirtoglou, Christoph Paul Schimanski, Heja Goli, and Krzysztof Wnuk. “Taxonomic Trace Links - Rethinking Traceability and its Benefits.” submitted to *Journal of Systems and Software to the New Ideas and Trends track*, Jul 17, 2022.

Contribution Statement

We use the contributor role taxonomy [1] to write the authors’ contributions for each of the studies conducted in this thesis.

Chapter 2

Waleed Abdeen: Conceptualization, methodology, formal analysis investigation, and writing - original draft.

Xingru Chen: Conceptualization, methodology, formal analysis and investigation and writing - review & editing.

Michael Unterkalmsteiner: Conceptualization, methodology, validation, supervision, and writing - review & editing.

Chapter 3

Waleed Abdeen: Conceptualization, methodology, formal analysis, investigation, and writing - original draft.

Krzysztof Wnuk: Conceptualization, methodology, investigation, supervision, and writing - review & editing.

Michael Unterkalmsteiner: Conceptualization, methodology, formal analysis and investigation, supervision, project administration, funding acquisition, and writing - review & editing

Alexandros Chirtoglou: Conceptualization, formal analysis and writing - review & editing.

Chapter 4

Waleed Abdeen: Conceptualization, methodology, formal analysis, investigation, project administration, and writing - original draft.

Michael Unterkalmsteiner: Conceptualization, methodology, formal analysis and investigation, supervision, project administration, funding acquisition, writing - original draft, and writing - review & editing.

Alexandros Chirtoglou: Conceptualization, formal analysis, investigation, writing - original draft, and writing - review & editing.

Christoph Paul Schimansk: Conceptualization, formal analysis, investigation, writing - original draft, and writing - review & editing.

Heja Goli: Conceptualization, formal analysis, investigation, writing - original draft, and writing - review & editing.

Krzysztof Wnuk: Conceptualization, supervision, and writing - review & editing.

Related Papers

Michael Unterkalmsteiner and Waleed Abdeen. "A compendium and evaluation of taxonomy quality attributes." *Expert Systems*, 2022, e13098.

Contents

1	Introduction	11
1	Overview	11
2	Background	12
2.1	Requirements Engineering	12
2.2	Verification	13
2.3	Digital Assets	13
2.4	Distances	13
3	Research Gaps and Methodology	14
3.1	Research Methodology	15
3.2	Threats to validity	19
4	Contribution	20
4.1	An approach for performance requirements verification and test environment generation	21
4.2	Challenges of Requirements Communication and Digital Assets Verification in Infrastructure Projects	22
4.3	Taxonomic Trace Links - Rethinking Traceability and its Benefits	23
4.4	Discussion	24
5	Conclusion and Future Work	28
2	PRO-TEST	31
1	Introduction	32
2	Background and related work	33
2.1	Software performance	33
2.2	Model-Based Testing	35
2.3	Related work	36
3	Research methodology	37
3.1	Systematic mapping study	39
3.2	Software requirements mining	43
3.3	Evaluating PRO-TEST	45

3.4	Threats to validity	45
4	Model-based performance testing	46
4.1	State of the art	46
4.2	Performance requirements in SRS documents	47
4.3	Discussion	48
4.4	Implications of the SMS on performance requirements in MBT	53
5	PRO-TEST	53
5.1	PRO-TEST approach development and description	54
5.2	Performance requirements model	57
5.3	Generating Test Environments	59
5.4	Example of PRO-TEST	60
5.5	Sample study - model evaluation	65
6	Discussion	68
6.1	Previous performance aspect classifications	68
6.2	Performance aspects inter-dependency	68
6.3	PRO-TEST benefits	69
6.4	PRO-TEST limitations	69
6.5	Observations on dependent and independent parameters	70
6.6	Performance prediction	70
7	Answering the research questions	70
8	Conclusions and future work	71
3	Challenges of Requirements Communication and Digital Assets Verification in Infrastructure Projects	73
1	Introduction	74
2	Related Work	75
3	Methodology	77
3.1	Data collection	78
3.2	Data analysis	79
3.3	Threats to validity	80
4	Results	81
4.1	Requirements validation	82
4.2	Requirements communication	84
4.3	Digital assets verification	87
5	Discussion	92
5.1	Customer-supplier communication	92
5.2	Challenges	93
5.3	Requirements communication	98
6	Conclusion	99
7	Acknowledgments	100

4	Taxonomic Trace Links	101
1	Introduction	101
2	Taxonomic Trace Links	103
2.1	Abstraction	104
2.2	Structure	104
2.3	Time	105
3	Research Methodology	105
3.1	Literature Survey	107
3.2	Validation study	108
3.3	Threats to Validity	111
4	Literature survey	111
4.1	Results	111
4.2	Discussion	116
4.3	Aspects and Quality Types	117
5	Validation Study	118
5.1	Association of Codes to Requirements	119
5.2	Verification of Codes in Design Models	120
5.3	Model verification based on traced requirements	125
5.4	Discussion	128
6	Conclusion and Future Work	128
A	Appendix	131
1	Included Papers in the SMS (Chapter 2)	131

CONTENTS

Introduction

1 Overview

Requirements engineering and verification (REV) play an essential role in software development. Requirements Engineering (RE) specifies what the software should do, and verification ensures that the software product is produced correctly. These two processes are currently distant [2, 3, 4]. According to Bjarnason et al., distance in software engineering (SE) "is a difference in position or level between entities that requires effort to traverse to accomplish a software development task" [3]. An entity could be an actor, artifact, or activity. An example of the distance between software engineering artifacts is the semantic differences between natural language (NL) requirements and test cases, where these artifacts are created by different individuals using different terminologies. These distances often contribute to communication flaws [2] between different roles and increased time for conducting software development activities [4]. Bjarnason et al. abstracted eight practices that are commonly used to reduce the distances in software engineering projects [3]. These practices are implemented on people, artifacts, or other aspects. An example is *aligning document structure and tracing*. In this practice, we establish trace links between requirements documents and test cases with different structures, which could impact multiple distances [3]. By reducing the time and effort required to access the relevant requirements to a specific test case, test engineers improve their understanding of the terminology used by requirements engineers as this information is easily accessible, thus reducing the semantic distance.

Two practices reported by Bjarnason et al. [3] are specifically relevant to reducing the distances between REV. 1) *Automated testing*: as software is being developed and changes are implemented due to added or changed requirements, tests need to be updated continuously to ensure maximum test coverage and high product quality. By using *automated testing*, we can get better test coverage [5]

since tests can be generated using algorithms that generate tests of all possible cases. Moreover, automated testing makes testing more efficient, and tests can be run without human action. 2) *Aligning document structure and tracing*: eases the execution and paves the way toward automation of other practices. Practitioners are facing challenges when implementing these practices [6], and a limited number of studies of aligning REV were published [7, 8].

In this thesis, we aim to investigate the use of *automated testing* and *aligning document structure and tracing* to reduce the distance between requirements engineering and verification in software and system engineering projects. In *automated tracing*, we study model-based testing (MBT), a technique that automates testing by generating test cases through modeling the system behavior [9, 10]. In *aligning document structure and tracing* we investigate using trace links to connect artifacts. In this thesis, we use the name of the specific technique model-based testing (MBT) and traceability to refer to these two practices. We argue that NL requirements written for software and system engineering projects are similar in their nature, as they represent the customer desire in the form of sentences. Furthermore, automated techniques that extract information from artifacts are concerned with language rather than the particular application domain [11].

By investigating the above practices, we understand how they reduce the distances between requirements engineering and verification. The remainder of the thesis is organized as follows. We present background information on requirements engineering, verification and distances in Section 2. Then we present the research gaps that we identified and the methodologies used in our studies in Section 3. Section 4 contains an overview of the studies included in this paper and the contributions of the thesis. We present our conclusion and planned future work in Section 5.

2 Background

2.1 Requirements Engineering

Requirements Engineering (RE) concerns the elicitation, identification, specification, and maintenance of the customer needs. The output of this process is a set of requirements documented in a textual or non-textual format (e.g., diagrams), and referred to as requirements specification. Practitioners tend to give low priority to requirements engineering [12]. We argue that this is because the value of requirements is not seen beyond telling the developer what is their list of tasks of the day. Researchers and practitioners [12] sees that the requirements specifications should not be neglected, and we should enable their use in the downstream project stages (e.g., design and testing).

In this thesis, we investigate both software and system requirements, as they are similar in terms of how they are documented. The majority of requirements are

written in natural language (NL) in software [12, 13, 14] and system [14] projects. Moreover, researchers have investigated and reported requirements challenges in both disciplines in the same study [15, 6].

2.2 Verification

According to SWEDBok [16] and SEBOK [17], verification is the process of ensuring the product is correctly developed. Verification activities are conducted throughout the development life cycle, such as verification of requirements, design models, and the end product. For example, in software engineering during verification of the source code through code reviews, the reviewer reads the code to find defects [18]. A similar process exists in system engineering and is referred to as inspection [17], where engineers examine objects of the system for visual or dimensional irregularities.

In this thesis, we refer to the digital artifacts whose quality is ensured during the verification process, as *verification artifacts*. In the previous example, the verification artifact in code review is the source code and in model checking it is the design model.

2.3 Digital Assets

According to Toygar et al. [19], digital assets are defined as any digital file (textual, images, audio, or video) stored on any electronic device (e.g., computer, mobile phone, or cloud) with the right to own the file

In this thesis, we studied digital assets in the context of system projects, where digital assets are mainly Computer-Aided Design models (CAD), Building Information Models (BIM), and System Information Models (SIM) [20].

2.4 Distances

The theory of distances in software engineering was proposed by Bjarnason et al. [3]. The theory gives an explanation for the causes of insufficient coordination and communication in software projects, and states that *distance* is a variation in a physical location, theoretical knowledge, or effort between entities in the software development processes. *An entity* could be an actor, artifact, or activity. These distances can be *changed* by implementing *software development practices*. There are eight distances in software engineering that fall under three categories [3]:

1. Actors: the distances between individuals which include *D1 geographical*, *D2 organizational*, *D3 psychological*, and *D4 cognitive* distances.
2. Artifacts: the variation between artifacts content, i.e. *D5 adherence*, *D6 semantic*, and *D8 navigational*. These distances affect the content of artifacts.

3. Activities: the variation in time between the execution of software activities (*D8 temporal*).

These distances can be affected by introducing practices that are relevant to actors, artifacts, or other practices. Karahappa et al. [8] extracted studies that investigated the requirements engineering and software testing alignment practices and mapped them into eight categories. The most frequently used practice is test generation from requirements specification, followed by improved traceability, and verification and validation in requirements engineering. In practice, MBT and traceability are the most used activities to align requirements engineering, and testing (RET) [6, 3].

3 Research Gaps and Methodology

The goal of this thesis is to investigate MBT and traceability w.r.t their ability to reduce the distances between requirements engineering and verification. We have identified the following research gaps in these areas (related to objectives indicated by OB1, OB2, OB3 and OB4) :

Gap 1. MBT has been extensively researched in the context of functional testing. Dias Neto et al. conducted two literature reviews [21, 9]. Utting et al. created a taxonomy for MBT approaches [10]. These studies [9, 10] give a good overview about the process and approaches, but they lack the detail about quality testing (e.g., performance or security). Felderer et al. [22] investigated in more detail MBT in the context of security testing. However, to the best of our knowledge *model-based testing in the context of software performance had not been reviewed*. Thus, we conducted the study in Chapter 2 to understand how automated testing can be used to verify performance requirements. (Objective OB1).

Gap 2. Communication in software [23] and system [24] projects is a critical success factor. Researchers have investigated communication challenges in software [25] and system [26] projects. Also, many studies explored the challenges associated with requirements allocation [27], verification and validation [28], and system integration [29]. Thus, we identified the research gap concerning *the inter coordination and communication of these processes with a focus on requirements' usage in subsequent stages of large infrastructure projects had not been investigated*. To address this gap we have conducted an exploratory case study (Chapter 3) to explore the current state-of-practice in communication and coordination of requirements engineering and verification (Objective OB2).

Gap 3. The findings of the study reported in Chapter 3 revealed challenges associated with the requirements engineering and verification in system projects.

In particular, traceability between artifacts produced during the project (requirements and design models) was inadequate. Moreover, during the verification process, where requirements are needed to check the conforms of the produced digital assets to these requirements, engineers could not benefit from the existing abstract trace links. Hence, *traceability of requirements to verification artifacts is still a challenge* in the context we have studied. We conducted the study in Chapter 4 to align requirements structure and trace them to verification artifacts (Objective OB3).

In Table 1.1, we list the objectives and research questions of this thesis to fill the identified research gaps.

Table 1.1: Research Questions

Objective	Research Question
OB1: Understand how automated testing can be used to verify performance requirements.	RQ1: How software performance requirements are verified using automated testing?
OB2: Explore the current state-of-practice in communication and coordination of requirements engineering and verification.	RQ2: What are the current requirements engineering and verification challenges in large system projects?
OB3: Align requirements structure and trace them to verification artifacts.	RQ3: How can trace links be created between requirements and design models?

Figure 1.1 shows an overview of the studies conducted in this thesis and how they are connected. Each chapter represents a study, and is illustrated using a rectangle. The dotted rectangle refers to a practice to reduce the distance between requirements engineering and verification. The identified research gaps are presented in ellipses.

3.1 Research Methodology

Systematic Mapping Study (SMS) is a research method that is effective in getting an overview of the work done and identifying research gaps in a specific area. We usually identify existing literature, extract data from the identified studies, then map these studies based on the extracted data. Petersen et al. [30] propose guidelines for conducting a systematic mapping study, and it involves five main steps.

1. Study identification: two main methods are used to retrieve literature. a) Database search [31]: a search string is developed using keywords that origin from the research questions. b) Snowball sampling [32]: a set of key studies in the research topic are identified by the researchers, then the studies are

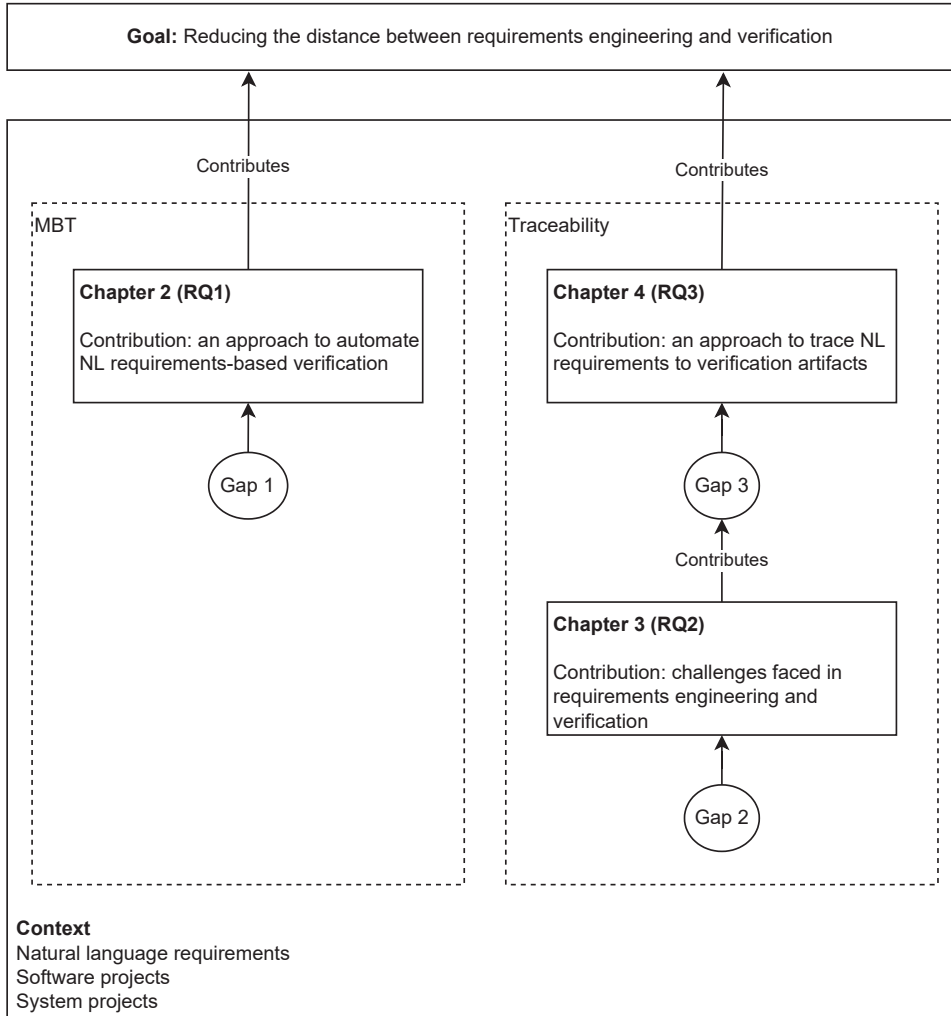


Figure 1.1: Roadmap of the licentiate thesis

collected using one or more iterations of forward and backward sampling based on the references of the key studies.

2. Study selection: selection criteria need to be identified to include only the studies that are relevant to the research topic. A set of inclusion and exclusion rules are identified to make a decision about the selection of the studies systematically. The researchers scan the identified studies, and make a decision using the selection criteria. A study may be excluded during later stages, e.g., during data extraction.
3. Data extraction: a set of data attributes are extracted from the selected studies. The selection of these attributes is guided by the research questions.
4. Data analysis: quantitative and qualitative results are synthesized, to answer the research questions.

In Chapter 3, our objective is to understand how automated testing can be used to verify performance requirements. None of the existing secondary studies review model-based performance testing. Thus, we choose to conduct an SMS, a good research method to get an overview of the work done in a research area, to fill this research gap.

Software requirements mining is the process of extracting data, such as the type of application (e.g., mobile or web) or requirement (e.g., FR or NFR) from the software requirements specifications (SRS). SRS mining is similar to mining software repositories (MSR) [33], where the source code in version control systems, the requirements in bug tracking systems, and communication between engineers are mined to extract (meta-)data.

Stol et al. [34] constructed a framework for software research that divides research into eight types. They frame MSR as a *sample study*, which is a study type that investigates the distribution of a population over a certain attribute [34]. The studied population could be people, where a questionnaire is sent out to a sample, or artifacts we extract data from (e.g., software repository mining).

In Chapter 2, one of the research questions asked *which aspects of performance requirements are used in MBT?*. Answering this research question required exploring the state-of-the-art and state-of-practice. We addressed the former by conducting an SMS and the latter by mining publicly available SRS documents¹, i.e. we conducted a sample study. Then, we developed an approach to verify performance requirements and generate test environments addressing the limitations of existing MBT approaches. Thereafter, we conducted another sample study on real project requirements to demonstrate the applicability of our approach.

¹A set of 77 software requirements specifications written for software and software-intensive projects.

Case Study is a research method to study a phenomenon in its natural setting [34]. A case study could be exploratory to discover the unknown or explanatory to give a reasoning for the studied phenomenon. Runeson et al. [35] advocate for an iterative process when conducting case studies. The data collection and analysis method are tuned as we are conducting the case study. For example, when conducting an exploratory study, the interview questions could be changed if we found that the collected data does not provide information to answer our research questions. The steps of the case study process are: 1) case study design, 2) preparation of data collection, 3) collecting evidence, 4) analysis of the collected data, 5) reporting.

In our research, we conduct empirical studies to characterize and solve problems that are relevant to the industry. We hypothesize that there are challenges related to requirements traceability in the context of large system engineering projects. Hence, our choice of case study as a research methodology in Chapter 3. We conducted two case studies in infrastructure projects. The infrastructure domain has a large set of well-defined requirements (i.e., regulations) that projects must adhere to.

Design Science is a research framework used to conduct and report research in an iterative manner [36, 37]. Hevner [38] has depicted design science as an iterative process which consist of relevance, rigor and design cycles. In the relevance cycle, the environments in which the problem is prevalent are identified. During the rigor cycle, the knowledge acquired during the research contributes to the strength of evidence. In the design cycle, the idea is grounded, refined and verified.

In Chapter 4 we seek knowledge about the use of indirection in traceability, and investigate the practical usage of taxonomic trace links. Thus, we chose to frame our research as design science. In our study, we conducted one iteration and in the future conduct more iterations to further refine and validate our solution

Validation Study is a study where intervention is applied to a problem to see if the intervention is feasible in practice and understand the practical challenges. The validation study should not be interpreted as a method that evaluates a solution, but rather as a method to discover any major design shortcoming in the intervention; these shortcoming are used to guide improvements to the solution.

In Chapter 4, we describe and further evolve the idea of taxonomic trace links that builds upon existing work [39]. Before investing resources in developing a fully working solution, we piloted the solution with practitioners on a set of general and project-specific requirements from the infrastructure domain. We created trace links from the requirements to design models and used these trace links to verify the conformance of the design models to the traced requirements.

3.2 Threats to validity

Threats to validity are present in research planning, execution, and reporting. There are four main validity threats affecting a research method: construct, internal and external validity, and reliability. We use the framework of Runeson et al. [40] to discuss the validity of our research.

Construct Validity regards the degree the research is designed and executed to avoid invalid results.

In the three studies (Chapters 2, 3, and 4), we have constructed research protocols following guidelines for research in software engineering [31, 40, 30, 34], which is rooted in the research questions for the corresponding study. Each protocol was prepared in an iterative manner before and during the execution of the research with input from at least two of the involved researchers.

In Chapter 2, some studies could be missed during the data collection of the SMS. We validated the recall of our search string on a set of studies reported in two top venues in the research area. The interview questions may not collect the required data to answer our research questions in Chapter 3. To mitigate this threat, we adapted the research questions to each interviewee's role. In Chapter 4, data triangulation was implemented to retrieve studies in the literature review from two sources. We used the studies from a systematic literature survey by Charalampidou et al. [41] and used snowball sampling to retrieve more studies. Furthermore, multiple researchers were involved in creating trace links in the validation study.

Internal Validity refers to the degree the research is designed and conducted to prevent systematic errors by researchers.

During data extraction of the SMS and SRS mining (Chapter 2), there is a significant researcher bias. We tried to mitigate this threat by involving two researchers during the data extraction phase. In the case study (Chapter 3), the semi-structured interview questions could have let the researchers ask questions that would make the results biased. We involved multiple researchers in each interview to mitigate this threat.

External Validity concerns with the representation and generalization of the results to the defined population.

In Chapter 2, there are threats that could hinder the generalizability of the study. First, in the SMS we could have missed some studies when we collected data using database search. There are inherited threats as well from the study by Dias Neto et al. [9] which we reused and extended the search string from. The borrowed search string from their study may not have returned all papers investigating MBT. We mitigated this threat by verifying the recall of the borrowed search string and then adjusting it. Second, the SRS collection that we used for the sample study may be outdated and does not include all domains of software and software-intensive

products (e.g., mobile application and autonomous cars). The list of performance aspects that we looked at is not exhaustive, but rather a representation of the most used aspects.

In Chapter 3, we have conducted only two case studies. It can be argued that the number of cases is not enough to generalize. However, it is possible to generalize from a single case provided it is representative of the population, and the focus is on analytical rather than statistical generalization [42]. We had two cases; each case represents a large infrastructure project; one was in its final building stages, while the other was in the first stages at the time of conducting our research.

The sampled requirements in the validation study of Chapter 4 introduce a threat to the generalizability of the results to the software domain. The sample is a set of 27 generic and project-specific requirements written in NL from an infrastructure project. Although the majority of requirements are written using NL in the system and software domain, some properties could be different for the requirements from different domains (e.g., structure and terminology), thus affecting the generalizability of the software domain. Another threat to the external validity is that the verification process of requirements in the product could be different between both domains.

Reliability regards how well the results are trusted. Another researcher should be able to reach the same results and conclusion as the authors did.

We mitigated this threat in Chapter 2 by 1) publishing the instruments of the SMS (study identifications and selection and data extraction and analysis), 2) we published the selection criteria, coding mechanism, and SRS collection of the SRS mining.

In Chapter 3, we present the protocol of our study and include the setup of our studied cases, the clusters of the interview questions, the interview steps, and the coding mechanism of the interview data.

In Chapter 4, we mitigated this threat by presenting the study selection and data extraction steps of the literature survey. Furthermore, in the validation study, we explain the properties of the sample requirements, the steps for conducting the validation study, and the experience of the involved researchers and domain experts.

4 Contribution

In this section, we give an overview of the studies that we conducted in this thesis. We start by presenting the contribution of each study. Then we synthesize the results of these studies and show how they contribute to the goal of our research.

4.1 An approach for performance requirements verification and test environment generation

Model-based testing (MBT) was adopted by practitioners [43] and reviewed by researchers [9]. However, the use of MBT to verify aspects of software quality had not been researched extensively [10] and MBT to verify software performance had never been reviewed to the best of our knowledge. Our aim in this study is to look for MBT models that verify the most relevant (in academia and industry) software performance aspects. We mapped the literature and mined performance requirements in a collection of Software Requirements Specifications. We have identified 77 primary studies from the literature and presented the results in three maps based on performance aspects, testing level, model, research method, study type, and contribution. We extracted 149 performance requirements and mapped them based on the performance aspect that they specify and the application type.

We had two main observations from our results. First, in both literature and practice, time-behavior was the most studied (in 62 papers) and specified (in 71 requirements) performance aspect compared to the other aspects. The other performance aspects were studied and specified to a lesser extent. We argue that requirements engineers and testers should pay more attention to other performance aspects (resource utilization, capacity, speed/throughput, and efficiency) as they have an impact on the performance perceived by the user of the software.

Second, although existing model-based performance testing techniques could successfully (semi-)automate the generation of performance tests from a performance model, these techniques miss out on the opportunities that MBT brings: 1) requirements validation, 2) check of the testability of requirements, and 3) the generation of an efficient test suite. The shortcomings of these techniques signal a need for a model that achieves these goals.

We developed PRO-TEST, Performance Requirements verificatiOn and Test EnvironmentS generaTion to address shortcomings requirements validation and requirements testability. PRO-TEST is an approach that verifies performance requirements and generates test environments using the performance requirements model. We depict our view of model-based performance testing in Figure 1.2. The grey boxes show our modification to the MBT process presented by Utting et al. [10]. The PRO-TEST process starts with performance requirements modeling (step 1-2) and ends with the generation of test environments specifications (step 6-1). The result of this process is 1) a more complete, quantifiable list of performance requirements and 2) test environment specifications to build test environments to run performance tests.

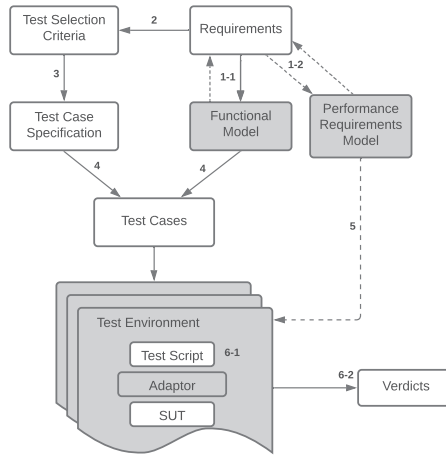


Figure 1.2: MBT process in the context of software performance [44]

4.2 Challenges of Requirements Communication and Digital Assets Verification in Infrastructure Projects

Researchers have explored the challenges in infrastructure projects [29, 28, 27]. These studies explored the challenges in individual stages (e.g., verification and validation [28] or communication) of the project. However, to the best of our knowledge, the inter coordination and communication of these processes with a focus on requirements' usage in subsequent stages of large infrastructure projects had not been explored.

We conducted two case studies and interviewed ten practitioners with expertise in requirements engineering, infrastructure design, and project management. We asked questions about the following processes: requirements validation, requirements communication and digital assets verification. We chose to focus on these process as they are the basis of communication between the customer and supplier, especially when projects are outsourced where multiple connection points exist between the client and the supplier.

We draw a per case process diagram for each of the mentioned processes and identified 14 challenges, their causes, and consequences. We mapped these challenges to four main clusters: project management, common RE, requirements quality, and trace links. These results support our hypothesis that there are challenges associated with traceability, which are: a) *granularity of traces*: existing trace links connect requirements to digital assets on a high abstraction level. Although the requirements can be specific, however, they are connected to the digital assets on the model level rather than the objects. b) *Verifying all requirements*: verifying all

relevant requirements in the design model is not feasible. The project has a long list of requirements that are implemented in multiple places in a model, and the process depends on the engineer's experience to find the relevant requirements and verify those requirements in the model.

4.3 Taxonomic Trace Links - Rethinking Traceability and its Benefits

As seen in the interview study (Chapter 3), traceability is still a challenge in large projects that must adhere to a large set of requirements. While the benefits of traceability are recognized, the cost of establishing and maintaining traceability seems to outweigh the expected benefits. In this study, we presented a traceability approach, namely Taxonomic Trace Links (TTL), surveyed the requirements traceability literature to show the novelty of our approach, and conducted a validation study with practitioners to show the applicability of our approach in practice.

TTL is an approach that connects a source and target artifact through the use of a third auxiliary artifact. An auxiliary artifact could be any knowledge structure (e.g., ontology or concept map). In our approach, we use a taxonomy. We show in Figure 1.3 traditional trace links and TTL. We argue that TTL addresses three main issues in current traceability approaches.

1. Abstraction: traced source and target artifacts of different types could have different abstraction levels, the third auxiliary artifact act as an adapter to connect these two artifacts. E.g., product-level requirements usually specify a certain functionality or feature, while tests that verify these requirements could be written as a unit, integration, or system test.
2. Structure: artifacts used in the software development process have different structures, apparent in outsourced projects where engineers in different organizations use different tools and techniques. A taxonomy acts as an interface that the tools adhere to.
3. Time: creating a direct trace link between two artifacts requires the existence of both of them. Introducing this indirection in the trace link through an auxiliary artifact makes it possible to initiate trace link creation early in the process. E.g., we can create trace links from the requirements to the auxiliary artifact without having any code or test written.

We conducted a validation study where we identified three themes of lessons learned related to requirements quality, taxonomy quality, and the alignment of the requirements and taxonomy.

In our literature survey we extracted data about the auxiliary artifacts and their usages in the traceability problem. We identified approximately 45 auxiliary artifacts with 13 usages in the requirements traceability process. Five auxiliary

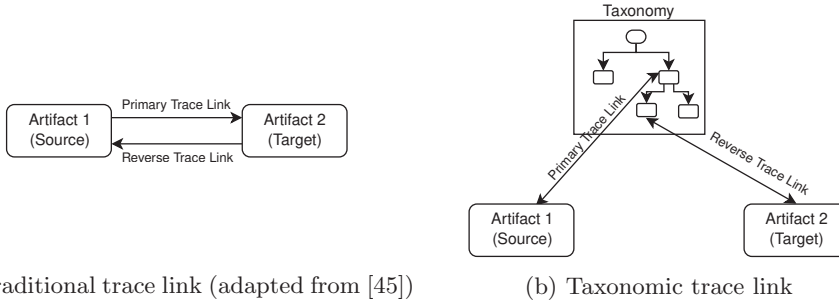


Figure 1.3: Traditional vs. taxonomic trace links

artifacts (taxonomy, ontology, early aspects, code aspects, and quality types) were used to classify the traced artifacts, similar to the usage of taxonomy in our approach.

Auxiliary artifacts have been researched to establish requirements traceability through classification. However, the problems that taxonomic trace links solve (abstraction, structure, and time) were not investigated by these techniques. They use auxiliary artifacts to: 1) recover trace links late in the development process, which may cause traceability not to be implemented [46]; 2) classify and trace artifacts of the same type [47, 48, 49], which does not tackle the problems of the different abstraction levels and structure of the traced artifacts; and 3) classify artifacts based on quality aspects [50, 51], and consequently creates abstract trace links.

4.4 Discussion

Model-Based Testing

MBT is used as a practice to close the gap between requirements engineering and testing (verification and validation) [3]. However, this usage was not manifested in the context of software performance, as we see from the results of (Chapter 2), which could be due to the following:

1. Performance testing is inherently different from functional testing, and the MBT process does not generalize to performance testing (and consequently NFR) [10] without adaptation. As we depicted in Figure 1.2, performance requirements modeling is different from functional requirements modeling, as a functional requirement specifies a certain functionality that the software should have, while a performance requirement specifies the desired quality attribute of a function or the whole system.
2. In some cases, performance models in model-based performance testing tech-

niques stem from the software design or code without considering the software requirements. As a result, the generated tests verify that the software works rather than the conformance of the software to the requirements.

We argue that PRO-TEST addresses these issues by modeling performance requirements and generating test environment specifications from the produced models. As we do not generate functional test cases, we rely on the integration of PRO-TEST in existing test generation processes.

Traceability

In Chapter 4, we chose to find solutions to trace links challenges. Traceability, as mentioned by Bjarnason et al. [3] reduces multiple distances between entities in general and requirements engineering testing in particular. Furthermore, traceability can be beneficial for change impact analysis. A product manager can make an informed decision about how a change in requirements will propagate in the software [52]. The challenges identified in Chapter 3, that are relevant to trace links identified two challenges: a) *granularity of traces*: existing trace links connect requirements to digital assets on a high abstraction level, and b) *verifying all requirements*: verifying all relevant requirements in the design model is not feasible as the process is time-consuming and depends on the engineer's experience to find the relevant requirements and verify those requirements in the model.

Implementing traceability in practice is challenging, as observed in several studies and summarized by Bouillon et al. [53]. One challenge is that the creator and user of trace links are seldom the same [54, 53], which is a common disadvantage [55] for both approaches of traceability; trace capture and trace recovery [45]. This conundrum motivated us to rethink how and, in particular, *when* trace links should be created and how we can design a trace process that is beneficial for the creators of trace links.

TTL (taxonomic trace links) introduce traceability between NL requirements and verification artifacts (Chapter 4). The introduced traceability makes it possible to retrieve the relevant requirements in a specific context during the verification process, which in theory should support engineers with *verifying all requirements*. Furthermore, using a hierarchical knowledge organisation structure, trace links can be created on different abstraction levels. This addresses the problem of trace granularity.

We saw from our validation study that it is possible to implement taxonomic trace links in practice. We were able to classify and trace requirements during the verification process. Establishing reliable traceability through classification is dependent on the quality of the traced artifacts, the granularity and uniqueness of the taxonomy, and the proper association of codes to the traced artifacts. We take these lessons to understand what process aspects could be tuned to improve our solution.

Impact on Distances

In Table 1.2 we map the practices that we investigate (MBT and Traceability) to the distances reported by Bjarnason [3]. The symbol (X) in the table means that the practice addresses the corresponding distance.

D1: Geographical. None of the investigated practices has a direct impact on the physical distances between requirements engineering and the verification process. We see this issue as relevant to the global software engineering area [2].

D2: Organizational. Introducing traceability between requirements and verification artifacts facilitates the discussion between engineers from the requirements and verification department. For example, during the design phase of an infrastructure project, when *a change in requirements* is introduced, requirements engineers from the client need to communicate with designers and quality engineers to assess the impact of this change. Requirements traceability within requirements and between requirements and other artifacts ensures that engineers are on the same page when communicating the impact of the change in their perspective department.

D3: Psychological. MBT and traceability impact psychological distances between requirements and test engineers, who have different perspectives on the software product. Both practices connect artifacts from both processes as follows: 1) MBT through the transformation of requirements into tests, this transformation has the effect that tests are connected with requirements. 2) Traceability by introducing trace links between artifacts from both processes. Thus, verifying all relevant requirements becomes easier.

D4: Cognitive. Cognition is essential in software engineering activities [56], where the tasks requires extensive knowledge and reasoning skills [56]. Humans have different levels of cognitive skills. A verification engineer needs to discover the errors in the product, while a requirements engineer needs to elicit stakeholders' needs. Consequently, their knowledge is different. TTL connect requirements and verification artifacts using domain-specific taxonomy, which harmonizes the domain knowledge of the engineers.

In our previous example of *a change in requirements*, introducing TTL, which have semantic information, can support change impact analysis. In the validation study of Chapter 4, it was possible to retrieve a relevant list of requirements during the verification process and check these requirements in the design models, and our knowledge about the relevant requirements expanded. Verification engineers can communicate and discuss the verification results, e.g., an abstract requirement could not be verified, with the requirements engineers to understand the requirement better or negotiate it.

D5: Adherence. With the incremental changes in the software product, the software may become less adherent to the written documentation. This phenomenon is known as entropy asset degradation [57, 58, 59]. In software, requirements specifications are the main source of documentation, and adherence of the software to this documentation can be increased by continuous verification of this documentation in the software product. MBT introduces automation in the verification process. When a *change in requirements* is introduced, the requirements models are changed as well to reflect the new state of the requirements. Then the automated generation of test cases or environments from these models is triggered. We were able with PRO-TEST to automate the process of generating test environment specifications from the created performance requirements models. Furthermore, traceable requirements reduce the verification time by easing access to relevant requirements to verify the product and relevant tests that should be run.

D6: Semantic. The taxonomic trace links (TTL) have the properties of a semantic link, which is a trace link coupled with terminologies that adds meaning to the link. Using taxonomy to classify traced artifacts unifies the terminology used in the produced artifacts during the software development. Requirements and test engineers could produce artifacts related to their processes while using similar terminology to the one used in the taxonomy.

D7: Navigational. The navigation between artifacts without existing traces is labor-intensive, and we risk not retrieving all documents in a specific context. The creation of explicit trace links between requirements and verification artifacts makes navigation between them easy, which otherwise could be difficult as we need to re-engineer the process every time we need to find the requirements relevant to specific test cases. The TTL approach has the advantage of creating trace links on different abstraction levels, which makes tracing a high-level requirements to individual test cases possible. A taxonomy is a set of classes organized in a hierarchy. A high level requirement can be classified with a class **A** while a more concrete test cases could be classified with the children classes **A1** and **A2**. We can retrieve the relevant test cases to the requirement by looking at the test cases with similar classification **A** or child classes **A1** or **A2**.

D8: Temporal. In software development, artifacts are created at different stages and times. MBT can generate tests early, after the requirements are specified, which brings the verification early, closer to the RE stage. One advantage of TTL is that it reduces the time between, when the artifact is created, and when the trace links are generated. Thus, supports the use of requirements in verification earlier.

By generating tests early from the requirements (MBT) and connecting requirements to the tests (TTL), it becomes easier to use test-driven development (TDD).

Table 1.2: Impact on the distances in software engineering

Distance	MBT	Traceability
D1: Geographical		
D2: Organizational		X
D3: Psychological	X	X
D4: Cognitive		X
D5: Adherence	X	X
D6: Semantic		X
D7: Navigational		X
D8: Temporal	X	X

A common testing techniques used in agile development model, were tests are written before the code and then the code is written in small increments until the tests passes [60].

5 Conclusion and Future Work

Conclusion. We have investigated MBT and Traceability practices to reduce the distance between requirements engineering and verification. We conducted an SMS and mined SRS documents to automate the generation of test artifacts from NL requirements by using a requirement model (Chapter 2). We explored the challenges faced between the customer and supplier during the exchange of artifacts in an infrastructure project through an interview study in two cases (Chapter 3). We proposed the taxonomic trace links approach and studied the feasibility of tracing and verifying project artifacts (NL requirements and design models), and discussed the differences between our approach and other traceability approaches by conducting a literature survey (Chapter 4).

MBT can be used to automate testing and reduce the distance between requirements engineering and verification. However, not all MBT techniques has the ability to reduce this distance effectively. MBT is most effective in reducing the distances between requirements and verification when the model used is built from the software requirements.

Traceability is essential in easing access to relevant information when needed and should not be seen as an overhead. The benefits that traceability brings extend beyond linking objects, as it opens new possibilities to support other activities in software and system projects (e.g., requirements-based verification). In order to increase the effectiveness of traceability in reducing the distance between requirements and verification, when creating trace links, we need to consider the difference in the abstraction, structure, and time between the linked artifacts

Future work. Our plan is to continue researching traceability between NL requirements and verification artifacts. In particular, we aim to develop and assess a solution that supports the (semi-)automated generation of taxonomic trace links between requirements engineering artifacts and verification artifacts. The planned studies are:

1. Develop a recommender system to classify requirements using domain-specific taxonomies and conduct experiments to evaluate its performance. The core research questions here is:

How does the taxonomic trace links recommender perform compared to other trace links recommender systems?

2. Conduct experiments to evaluate the effect of taxonomy on the performance of the recommender system. The core research question here is:

To what extent does the quality of the taxonomy used for classification affect the performance of the recommender system?

3. Adapt taxonomic trace links recommender to classify other software artifacts (e.g., test code), and evaluate the recommender performance by experiments. The core research questions here are:

What properties from a test code are required to classify it using domain-specific taxonomies?

How does the taxonomic trace links recommender perform compared to other trace links recommender systems ?

4. Explore the practical use cases of taxonomic trace links in practices by conducting a case study. One possible use case for taxonomic trace links is requirements-based verification. The core research question here is:

How effective is the taxonomic trace link recommender in supporting requirements-based verification in practice?

At the end, we expect to have a comprehensive solution to trace NL requirements to verification artifacts, potentially supporting practitioners to implement traceability in their software projects with acceptable effort.

An Approach for Performance Requirements Verification and Test Environments Generation

Background: Model-Based Testing (MBT) is a method that supports the design and execution of test cases by models that specify the intended behaviors of a system under test. *Motivation:* While systematic literature reviews on MBT in general exist, the state-of-the-art on modeling and testing performance requirements has seen much less attention. *Method:* Therefore, we conducted a systematic mapping study on model-based performance testing. Then, we studied natural language software requirements specifications in order to understand which and how performance requirements are typically specified. Since none of the identified MBT techniques supported a major benefit of modeling, namely identifying faults in requirements specifications, we developed the Performance Requirements verification and Test Environments generation approach (PRO-TEST). Finally, we evaluated PRO-TEST on 149 requirements specifications. *Results:* We found and analyzed 57 primary studies from the systematic mapping study, and extracted 50 performance requirements models. However, those models don't achieve the goals of MBT, which are validating requirements, ensuring their testability, and generating the minimum required test cases. We analyzed 77 Software Requirements Specification (SRS) documents, extracted 149 performance requirements from those SRS, and illustrate that with PRO-TEST we can model performance requirements, find issues in those requirements and detect missing ones. We detected three not-quantifiable requirements, 43 not-quantified requirements, and 180 underspecified parameters in the 149 modeled performance requirements. Furthermore, we generated 96 test environments from those models. *Conclusion:* By modeling performance requirements with PRO-TEST, we can identify issues in the requirements related to their ambiguity, measurability, and completeness. Additionally, it allows to generate parameters for test environments.

1 Introduction

Performance aspects such as time behavior, capacity, or throughput, are essential non-functional requirements (NFR) of software products. Performance testing is the process of measuring the availability, response time, throughput, and resource utilization of a software product [61]. The importance of software performance and relation to functional requirements is acknowledged since the 1990s [62]. A real-world example is HealthCare.gov, a "health insurance exchange website" run by the United States government, where on the launch day 99% of people who wanted to get insurance failed to register [63]. Further investigations showed that no adequate performance testing was performed [64].

Performance-related issues can have a large impact on cost, especially if those issues are not treated early [65, 66, 67]. Another example of a software performance issue was Pokemon Go [68], a mobile game that, after the initial roll-out, became unusable in many countries. The large number of users caused server failures, leading to a delayed roll-out of the game to reduce the load [68]. A potential reason for such a failure is the different nature of performance requirements compared to functional requirements, which makes it difficult for developers to translate performance requirements into written code [69]. Therefore, performance testing is necessary, since it can detect the causes of performance-related issues and verify whether the software product meets the requirements or not [69].

Model-based testing (MBT) is a software testing approach that uses an abstraction of the system (or part thereof) to generate test cases [10]. According to a software testing survey conducted in Canada [43], more than 35% of the respondents use MBT approaches to generate test cases in their projects. This indicates that MBT is prevalent in the industry. MBT forces testability into the product design when creating the model. The model is created from the requirements and describes the behavior of the system. Successfully modeled system requirements indicate that those requirements are testable, complete, and can be validated since they were formalized in an unambiguous manner [70].

Many studies explored the state-of-the-art of MBT [21, 9, 22, 71, 10, 72]. Utting et al. [10, 72] created a taxonomy of existing MBT approaches and tools, and Dias-Neto et al. [21, 9] systematically reviewed the literature of MBT in 2007 and 2010. These studies agreed that the existing MBT approaches focus on testing the functional rather than the non-functional part (i.e., quality aspects) of the system. Later, Häser et al. [71] reviewed the literature for model-based integration testing for NFR, and Felderer et al. [22] model-based security testing. A look at the state-of-the-art for model-based performance testing is missing.

In this paper, we study the current status of model-based performance testing, and identify approaches that we can use to model different aspects of performance requirements. Then, we propose the Performance Requirements verification and Test EnvironmentS generation approach (PRO-TEST) which supports model-

based performance testing by checking the ambiguity, measurability, and completeness of performance requirements, and generating test environments. Finally, we evaluate PRO-TEST on real software requirements specifications.

The main contributions of this study are:

1. A categorization of MBT studies in the context of performance requirements, based on the performance aspect, testing level, study type, research method, model type, application type, and contribution.
2. A categorization of the Software Requirements Specifications (SRS) from a public repository [73], based on the described application type and performance requirements.
3. PRO-TEST, an approach to model performance requirements to verify them, understand what should be tested, and generate test environments.
4. An evaluation of PRO-TEST, illustrating its benefits and drawbacks.

The remainder of this paper is organized as follows. Section 2 introduces the concepts of software performance and model-based testing, and reviews related work. Section 3 illustrates the design and methodology used in our research and the validity threats. In Section 4 we present state-of-the-art and state-of-practice of model-based performance testing. Section 5 presents PRO-TEST and the obtained benefits but also the faced challenges when modeling performance requirements. We discuss PRO-TEST in relation to literature in Section 6. Section 7 answers our research questions. Finally, we conclude the paper in Section 8 with directions for future work.

2 Background and related work

In this section, we briefly review aspects of software performance, model-based testing, and related work.

2.1 Software performance

Software performance is considered in many software quality models [74, 75]. Synthesizing these quality models, as shown in Table 2.1, the main aspects of software performance are time behavior [76, 77, 78], capacity [78], resource utilization [76, 77, 78], speed/throughput¹ [77] and efficiency [79, 76, 80, 81]. Next, we provide a definition of these software performance aspects.

¹The meaning of the symbol "/" is "or". We kept both words because they are both used frequently in performance.

Table 2.1: Quality models and their related performance aspects

Quality Model Name	Performance Aspect
McCall's	Execution Efficiency, Storage Efficiency
Bohem's	Accountability, Device Efficiency, Accessibility
Dromey's	Internal Efficiency, Descriptive Efficiency
FURPS	Speed, Efficiency, Availability, Accuracy, Throughput, Response Time, Recovery Time, Resource Usage
ISO9126	Time Behavior, Resource Utilization, Efficiency Compliance
ISO25010	Time Behavior, Resource Utilization, Capacity

Time Behavior: the time required to perform specific tasks or complete requests. It usually has multiple instances or values depending on different anticipated capacities (i.e., the number of users). This aspect is included in all three models (ISO9126, ISO25010, and FURPS) as time behavior or response time. It is an explicit aspect, that is used by the users to infer software performance. It could have a direct effect on the usability of the software.

Resource Utilization: the amount or percentage of the resources used to run the software. The software should not always utilize all resources when running, instead, it should be limited to a specific amount so that it has a margin for peak times and new updates that would require more resources.

Capacity: the maximum capacity (in terms of requests, sessions, users, data, etc.) that the system can handle without crashing. This aspect is crucial when planning for the project in later stages, especially when considering scalability. If not accounted for, it could result in an overload of the system, which would affect the business operations and lead to extra charges. Capacity gives an insight into the anticipated data size used by the software, which would affect the decision regarding the required resources for the system to operate.

Speed/Throughput: the number of requests or processes per time unit that the system can handle while still maintaining the time behavior requirements.

Efficiency: the relation between the output (i.e., time behavior, speed) and the input (i.e., capacity, resource utilization). This is a relatively complex aspect since it is affected by all other mentioned aspects of performance.

2.2 Model-Based Testing

Model-based testing is a software testing technique that automates the process of test case generation from a model that represents the system under test (SUT). MBT consists of three main tasks [82]: designing a functional test model, determining test generation criteria, and generating the tests. The model could be an end-to-end model, e.g., a business process or per function process model. Abstract test cases are generated from a systems' model by random generation, search-based algorithms, model-checking, symbolic execution, theorem proving, or constraint solving [10, 83]. Then a tool builds the test skeleton to test the software.

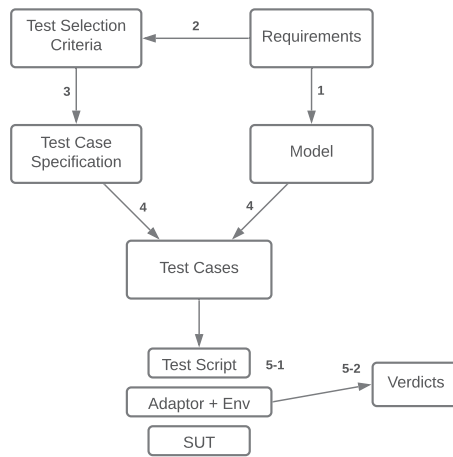


Figure 2.1: MBT Process Diagram from Utting et al. [10]

Utting et al. [10] present five steps of the MBT process. We illustrate this process in Figure 2.1. In *Step 1*, a test model is created from the requirements. The model can be either created specifically for testing or reuse some parts of the models used at the design phase. In the case of the latter, the test model should be independent of the design model, so issues in the design phase do not appear in the test model. A model should be verified with little effort to ensure the efficiency of the MBT approach. In *Step 2*, test selection criteria are defined, which will set the rules for the automatic generation of test cases. Examples of test selection criteria are system functionality (requirements-based), the structure of the test model, or properties of the environment. In *Step 3*, test case specifications are written as a more formal representation of test case selection criteria. In *Step 4*, the test specification is, with help of the model, transformed into concrete test cases. At this stage, algorithms are used to select the minimum set of test cases that ensure full test coverage. In *Step 5*, the tests are run on the SUT in a test environment. First, test inputs are fed to the function under test (5-1), then the test verdicts are

recorded by comparing the test results with the expected outcome (5-2).

There are many benefits associated with MBT. It has been shown to be effective in testing real-time adaptive systems [84], verifying the system behavior, and identifying possible performance enhancements. Furthermore, the benefits of MBT automation are generally more numerous the more testing the system requires [85]. Another benefit is that MBT finds missing and unclear requirements by modeling the systems' requirements [86, 87]. Besides, MBT can make the requirements more understandable for software engineers [69]. Since performance requirements are often written at a high abstraction level, it may be difficult to understand how they impact software design and code. This could be made easier by modeling functional and non-functional requirements using the same model. A UML activity diagram that models functional requirements could be annotated with performance requirements [88]. We could see in the resulting model where the performance requirements apply in the software.

2.3 Related work

There exist many studies that investigate MBT to test functional requirements, while fewer studies focus on non-functional requirements. Utting et al. created in 2006 [72] and 2012 [10] respectively a taxonomy for model-based testing to categorize the existing approaches and tools, as well as to classify their usefulness. Their study focused on functional requirements testing. In general, there is no clear distinction between functional and non-functional requirements when MBT is applied [89].

Although MBT for non-functional requirements is not explored extensively, there are still some studies in this area. A systematic review (78 papers) of MBT approaches by Dias-Neto et al. [21], published in 2007, was not limited to functional requirements and explores the non-functional aspects considered by the models. Some limitations of using MBT for non-functional requirements were pointed out by the study. The irregular behavior of software users makes it hard to create a behavioral model of non-functional requirements. Another challenge is the limited support for non-functional aspects in the existing MBT approaches; NFRs like usability, reliability, and security were not supported. Moreover, the majority of MBT approaches proposed by research are never used in industry [21].

Dias-Neto's original study was renewed in 2010 [9] (including 219 papers), with a focus on the techniques used for modeling, coverage, and the challenges of MBT. This study introduces selection criteria for MBT approaches based on their characteristics. The use of MBT techniques was still difficult, as observed in their previous study in 2007. Apparently, NFRs (usability, reliability, and security) that were not possible to test with MBT (according to the 2007 review [21]), started to get some attention in research. The difference between these studies [21, 9] and the systematic mapping study presented in Section 4 is that ours has a more narrow focus on model-based performance testing.

In 2014, Häser et al. [71] conducted a systematic literature review of model-based integration testing. They asked in their research questions about the software paradigms, assessment type, and which NFR can be tested with MBT approaches. However, they did not ask whether the MBT approach tests different aspects of an NFR (i.e., what aspects of performance were tested using these MBT approaches?), and they scoped their research to integration testing. Their findings indicate a lack of research in model-based integration testing for NFR.

In 2016, Felderer et al. [22] presented a taxonomy and systematic classification of model-based security testing. They extended the study of Dias-Neto et al. [9] while focusing on security requirements. Woodside et al. [69] described the domain of software performance engineering (SPE). They did a survey of current work on a sample of papers in SPE and pictured the future of SPE. They collected some models and methods which are used for performance and listed many benefits of modeling performance. The focus of that study is to provide a look at the future of model-based performance testing. In contrast, our study focuses on identifying current techniques that can be used in practice.

Motivated by this research gap, the lack of systematic reviews in MBT of performance requirements, the limited support for NFR in general, and performance in particular in existing techniques, we focus our research on finding and studying different performance requirements models, for the purpose of using them in MBT.

3 Research methodology

To achieve our research aim defined in Section 1, we have identified the following four objectives.

- **O1** Identify which aspects of performance are important and can be modeled.
- **O2** Identify modeling techniques and methods that suit performance requirements.
- **O3** Identify a modeling approach that can validate performance requirements, ensure that those requirements are testable, and support the generation of test cases, all three of which are key aspects of MBT.
- **O4** Evaluate the identified modeling approach on a set of requirements specifications.

In alignment with those objectives, we define our research questions in Table 2.2.

Figure 2.2 shows the steps of our research in alignment with the research questions. First, we start with a systematic mapping study (SMS). The mapping study is an appropriate method for gaining an overview of a particular research area. We explored which performance aspects were studied and modeled using MBT (RQ1,1,

Table 2.2: Research Questions

Number	Research Question	Purpose	Objective
RQ1	Which aspects of performance requirements are used in MBT?	There are many performance aspects, e.g., time, speed, and capacity, as explained in Section 2.1. Those aspects may have different ways of modeling and testing.	O1
RQ1.1	Which aspects of performance requirements have been studied?	Explore the studied aspects of performance requirements in MBT.	O1
RQ1.2	Which aspects of performance requirements can be modeled?	Explore the usage of MBT to model different aspects of performance requirements.	O1
RQ1.3	Which aspects of performance requirements are used in real-life projects?	Explore the performance aspects that are specified and relevant in real-life projects.	O1
RQ2	How to implement MBT on performance requirements aspects?	Explore the different MBT approaches that support the modeling of performance requirements to understand the current state of the art of MBT for performance requirements.	O2, O3
RQ2.1	What type of models can be used to model performance requirements aspects?	There are many models used in MBT. However, that does not mean all of them could be used to model all aspects of performance requirements.	O2
RQ2.2	Which performance requirements models achieve the goals of MBT?	Find models that achieve MBT goals, which are validating requirements, ensuring their testability and generating the minimum required test cases.	O3
RQ3	To what extent is the identified approach effective at modeling performance requirements written for real-life projects?	Evaluate the modeling approach that we identified in the previous step, to ensure its applicability on real-life projects with different aspects of performance requirements.	O4

RQ1.2), and what models exist to model performance requirements (RQ2.1). Second, we conducted a sample study on real-project requirements, for the purpose of finding out the relevance of performance aspects in practice (RQ1.3). Based on the results from the SMS and software requirement mining, we developed PRO-TEST (RQ2.2). Finally, we conducted a sample study, to evaluate PRO-TEST (RQ3). We focus our study on the domain of software-intensive systems.

A Systematic Literature Review (SLR) and an SMS are research methodologies that systemically survey the literature but differ in their aim, execution, and outcome [90, 91]. An SLR aims to aggregate data from the literature and has specific research questions for that purpose, while an SMS aims to explore trends and identify gaps in research. In terms of execution, an SLR requires a quality assessment to be conducted on the extracted papers, while it is not the case for an SMS. The

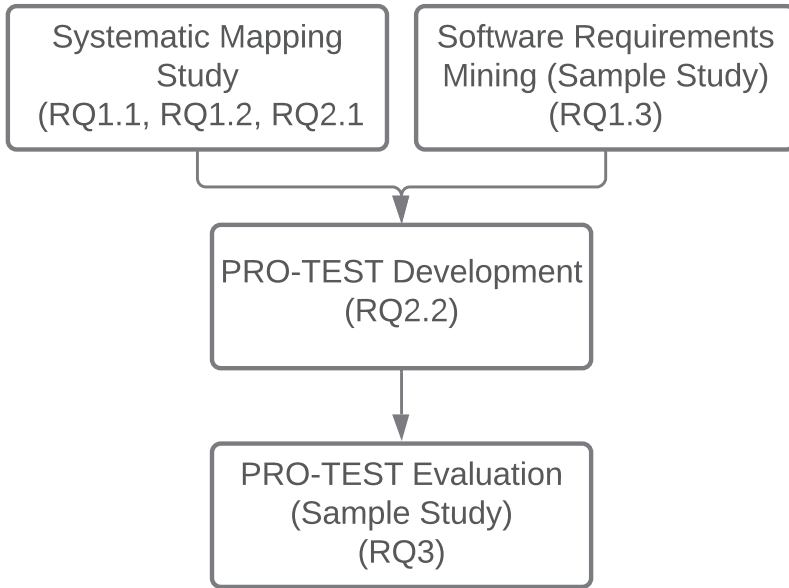


Figure 2.2: Research Methodologies

output of an SLR is a synthesis of the reviewed studies, while an SMS classifies a set of papers based on different dimensions.

A sample study is a form of research done on a sample of the population for generalization [34]. The data could be collected using interviews, questionnaires, metric reports, or available for access online, e.g., in a software repository. One of the research methods associated with sample studies is software repository mining [34]. Software repository mining research usually uses open-source software repositories. There is no human to collect data from, i.e., no interviews or questionnaires are involved.

The purpose of evaluating PRO-TEST is to validate that it works in practice, i.e., it can model the performance requirements and generate test environments. Similar to the software requirements mining approach described in Section 3.2, we conduct again a sample study, i.e., we use an openly accessible resource for software requirements specifications.

3.1 Systematic mapping study

We developed the SMS protocol based on the SLR conducted by Dias Neto et al. [9], following the guidelines by Petersen et al. [91]. There were two reasons for

choosing this study by Dias-Neto. First, the research group has conducted two SLRs [21, 9] on MBT using the same protocol. This provides some evidence for the repeatability of their study. Second, there was enough information presented about the search keywords and procedure, making it easier to adapt and extend the protocol. The choice of reusing and extending an existing protocol has however also disadvantages. The study of performance requirements concerns research beyond MBT, such as requirements engineering and software testing in general, software performance engineering and agile software development. Hence, we emphasize that our review covers the area of performance requirements within the scope of MBT only.

Study identification

Choosing the search strategy: We used keyword search in digital databases similar to the search method used by Dias Neto et al. 2010 [9]. They used six databases for their search. Two of the databases (i.e., Compendex IE and IN-SPEC) we did not have access to. Therefore we ran the search on the other four databases (SCOPUS, ACM, IEEE Xplore, and Web of Science). We searched the title, keywords, and abstract of the paper on SCOPUS, WoS, and ACM, while we searched the full text of IEEE (due to a limitation of the database).

Developing the search: We took the search string used by Dias Neto et al. [9] and extended it to fit the purpose of our research. The keywords we added are related to performance. We extracted those keywords from the quality models for software performance discussed in Section 2.1. Table 2.3 shows the borrowed search string and the extension with performance-related keywords.

Evaluating the search string: We evaluated the quality of the search string to mitigate the risk of missing key papers. We did that in two steps:

- We ran Dias Neto et al. [9] search string on the selected databases and randomly checked whether the returned research papers were presented by Dias Neto et al. in their study [9].
- To validate the whole search string including the extension, we reviewed the papers published at the *International Conference On Software Testing Verification And Validation (ICST)* over the period 2014-2018. We read the title and abstract to see if the topic is related to model-based performance testing. We collected the papers related to our topic and looked for them in our search results. We found three papers in the ICST conference proceedings that were not returned by our search string. After further analysis of the search string, we removed a part of Dias Neto et al. search string (*approach OR method OR methodology OR technique*) and adjusted our extension to ensure those papers are included.

Table 2.3: Search strings used in the SMS

Description	Keywords
Borrowed search string from Dias Neto et al.	((“model based test”) OR (“model based testing”) OR (“model driven test”) OR (“model driven testing”) OR (“specification based test”) OR (“specification based testing”) OR (“specification driven test”) OR (“specification driven testing”) OR (“use case based test”) OR (“use case based testing”) OR (“use case driven test”) OR (“use case driven testing”) OR (“uml based test”) OR (“uml based testing”) OR (“uml driven test”) OR (“uml driven testing”) OR (“requirement based test”) OR (“requirement based testing”) OR (“requirement driven test”) OR (“requirement driven testing”) OR (“finite state machine based test”) OR (“finite state machine based testing”) OR (“finite state machine driven test”) OR (“finite state machine driven testing”)) AND (software)
Extension	AND (performance OR efficiency OR capacity OR load OR speed OR responsiveness OR stability OR timing OR (“time behaviour”) OR (“time behavior”) OR (“response time”) OR (“response-time”) OR (“resource utilization”) OR (“resources utilization”) OR (“resource consumption”) OR (“resources consumption”) OR throughput OR throughput OR spike OR stress OR volume OR size OR scalability OR peak OR (“wait time”) OR latency OR delay OR workload OR (“concurrent users”) OR (“concurrent requests”))

Selection criteria

We developed the following inclusion and exclusion criteria.

Inclusion:

1. The publication is available in full text.
2. The publication language is English.
3. The date of the publication is within the range of August 2009 (the date when Dias Neto et al. [9] conducted their search) and February 2019 (when we conducted our search).
4. The publication proposes and/or evaluates model-based performance testing techniques.

Exclusion:

1. The publication presents secondary studies, i.e., SMS, SLR, literature survey.
2. The publication is not related to the topic model-based performance testing.
3. Duplicated publications that refer to the same study.

4. The publication is about model-based mutation testing
5. Proceeding, table of content, book, tutorial, demo, editorial

After careful analysis of the model-based mutation testing approach, we have decided to exclude it. Although it uses MBT as a basis, it is concerned with introducing faults during the test to find issues in the system rather than the modeling and test case generation.

Quality assessment

No detailed quality assessment was conducted. Since the goal of our SMS was to find a method that we can use, there was no need to evaluate the quality of each paper selected for our research.

Data extraction

We extracted the following data from our and Dias-Neto et al.'s [9] primary studies (after we applied our inclusion/exclusion criteria).

Performance aspect: We extracted data related to the five performance aspects discussed in Section 2.1, i.e., time behavior, resource utilization, capacity, throughput, and efficiency. We added a "not specified" category for those papers that do not mention or focus on a specific aspect of performance. This classification supports answering RQ1, RQ1.1, and RQ1.2.

Testing level: Testing can be conducted on five different levels [92]: acceptance, system, integration, module, and unit level. This classification supports answering RQ2 and determines on which level performance testing is conducted.

Study type: We used Stol et al. [34] to classify study types in software engineering: field study, field experiment, experimental simulation, laboratory experiment, judgment study, sample studies, formal theory, and computer simulation. This classification helps us to understand how mature the studied MBT techniques are, i.e., whether they are empirically studied and adopted by industry or initial proposals that require more empirical evidence. This is an additional criterion for choosing the model and answering RQ2, RQ2.1, and RQ2.2.

Research method: The research method differs from the study type. A research method defines the set of rules and practices to follow, having a specific goal in mind, i.e., answering a set of research questions. The study type is a grouping of different research methods based on their "metaphor, purpose and goals" [34]. In software engineering research, many research methods can be associated to study

types [34]. Some of those methods are case study, experiment, survey, and concept development. Since there is no complete list of those research methods, we kept this classification dynamic and extracted the options directly from the research papers. This classification helps to distinguish between papers that present a new approach or theory to others that empirically evaluate existing approaches.

Model type: We classified each paper based on the approach used to model performance requirements. The classification is based on the essence of the model, i.e., some models were novel while others were extensions of previous models. For example, Maâlej et al. [93] present timed-automata, while Abbors et al. [94] present a probabilistic extension of timed-automata. This helps to determine the frequency of the models used for performance requirements and answer RQ2.1. We did not have predetermined options for this classification, since one of our research objectives was to identify all possible modeling approaches.

Application type: We classify the type of the application (e.g., web application, mobile, desktop) to understand where model-based performance testing is used or studied. This is also a dynamic classification with no predetermined options.

Contribution: This classification assigns papers into categories based on their contribution to the field (e.g., tool, method, evaluation). With this classification, we can understand the maturity of the models.

Data analysis

We use the frequency of the extracted data, discussed in the previous section, to analyze the state-of-art in model-based performance testing.

Also, to identify a model that can achieve the MBT's goals, we examined the following aspects of the identified MBT techniques:

- reported benefits of modeling performance requirements
- modeled performance aspects
- type and strength of evaluation of the proposed method

3.2 Software requirements mining

The research questions RQ1 and RQ1.3 in Table 2.2 were answered by conducting software requirements mining. Ferrari et al. [73] published a data set [95] that contains a collection of software requirements specifications gathered from various industries and applications. There are 77 SRSs in total in the collection from which we constructed a subset as described next.

Selection criteria

Inclusion: the SRS and the individual requirements that are classified and shown in our results have the following properties.

- SRS: have at least one performance requirement.
- Requirement: fits in one of the descriptions for performance aspects in Section 2.1.

Exclusion: the SRS and the individual requirements that we excluded from our classification and the results have the following properties.

- SRS: without any performance requirements or not written for a software product.
- Requirement: does not fit in any of the performance aspects descriptions.

Coding

Since the data in the SRS documents is of qualitative nature, we used coding to efficiently identify and extract relevant information. The codes we created are based on having three dimensions (performance aspect, application type, and quantifiability) that we describe next.

Performance aspect: We extract five performance aspects, i.e., time behavior, resource utilization, capacity, speed/throughput, efficiency, and a general option for the performance requirements that did not fit in any of the five aspects' descriptions. We apply this classification to each performance requirement and provide thereby data to answer RQ1.3.

Application type: Similar to the SMS, we extract the type of application specified in the SRS, e.g., web application, mobile application, embedded system, etc. This allows us to evaluate whether the SRS data set is a good presentation of the population (i.e., software products).

Requirements quantifiability: Testability is one of the major criteria in requirements verification and validation [79]. The requirement "must be specific, unambiguous, and quantitative wherever possible" such that a developer can write software code that satisfies the requirements. The performance requirement should be quantitative and quantified to be testable. We evaluated each requirement by looking for numerical values.

3.3 Evaluating PRO-TEST

We evaluated PRO-TEST on a set of realistic software requirements specifications (SRS) containing performance requirements. The evaluation was done by modeling the performance requirements and assessing the quantifiability and degree of quantification of the specified requirements, and identifying the possible missing requirements.

3.4 Threats to validity

In the SMS there were threats related to the data extraction methods. 1) We may have missed some papers because two databases used by Dias Neto et al. [9] we did not have access to. To keep this to a minimum we made sure that we use the SCOPUS database, which includes publications from different technical publishers. 2) We may have excluded papers by our search string. We extended the search string from Dias Neto et al. [9] study with words related to performance. This could lead to fewer results if some keywords are missing from the search string. We tried to include as many keywords as possible and used performance checklists to make sure this threat is kept to a minimum. 3) Another type of threat is related to the human factor; we could have interpreted the data in the wrong way or placed a paper in the wrong classification. We addressed this threat by having the selection and classification done by two researchers independently and the results were then compared. When conflicts were discovered the corresponding paper was discussed by both researchers and if still no consensus could be achieved, a third researcher was consulted.

In software requirements mining, the human factor also introduces threats to validity. First, we could have coded some requirements wrongly or missed out on some performance requirements from the SRS documents. We mitigated this threat by having two researchers involved in coding. The researchers coded a sample of seven SRS documents independently and compared the results. When conflicts were discovered, the corresponding requirement was discussed by both researchers. Then we divided the work equally between the two researchers. When no consensus could be achieved by the two researchers a third researcher was consulted. Second, the sample size may not be enough for generalization since the SRS collection had 77 documents that might not cover all application types or represent the population, i.e., software products.

Finally, in the implementation of PRO-TEST, the small sample size is not enough to generalize the competence of the approach. Only 34 SRS documents of the SRS collection had performance requirements, which might lead to three issues: 1) The sample we chose might be small to represent the population, i.e., software products. 2) The SRS collection from Ferrari et al.'s study [73] might not be a good representation for the population as well. 3) The most recent SRS document goes back to 2010, which could be considered old. A validation of the

model on more recent SRS documents is required.

4 Model-based performance testing

This section reports on the results from the SMS on model-based performance testing (Section 4.1) and on the prevalence of performance requirements in a publicly available repository of software requirements specifications (Section 4.2). We discuss our findings in Section 4.3.

4.1 State of the art

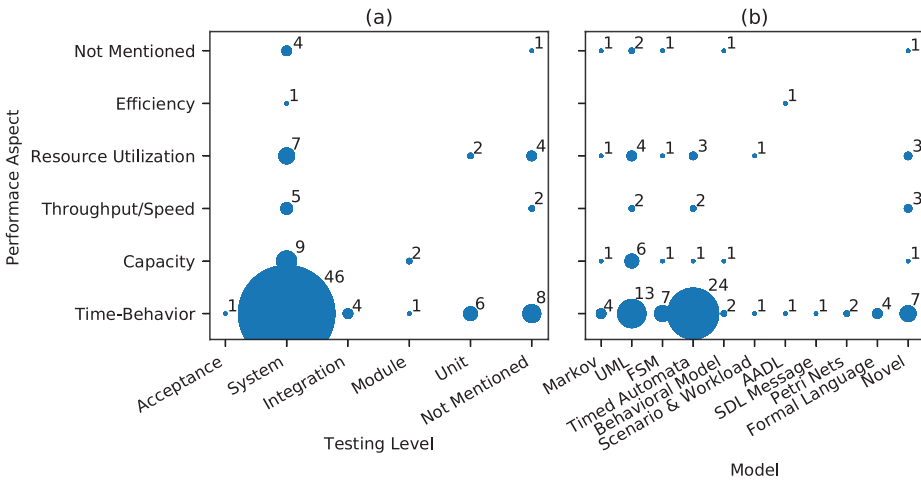


Figure 2.3: Papers mapping between a) performance aspect and testing level b) performance aspect and model

We identified 57 primary studies through our database search and extracted 20 from Dias-Neto’s study (see Appendix 1²). A paper could be mapped to more than one value in each classification, which depends on the content of the paper. The choice of these maps was driven by our research questions. We show in Figure 2.5 and Figure 2.3 the relation of the performance aspect with all other research area classifications. Moreover, a typical SMS should classify papers in both 1) the research area and 2) the research type [30], hence our choice of Figure 2.4.

²Additional materials including the list of primary studies, the mapping of papers to each classification, grouping of the models, data from Dias Neto’s study [9], the SRS collection, extracted performance requirements, the modeling of those requirements using PRO-TEST and the excluded performance requirements are available online [96].

In Figure 2.3 the y-axis represents the performance aspects, while the x-axis in Figure 2.3 (a) represents the testing level and in Figure 2.3 (b) the model types. The "Not mentioned" option in the performance aspects, represents the papers that did not mention or focus on any aspect. We categorized the extracted models based on the essence of the model.

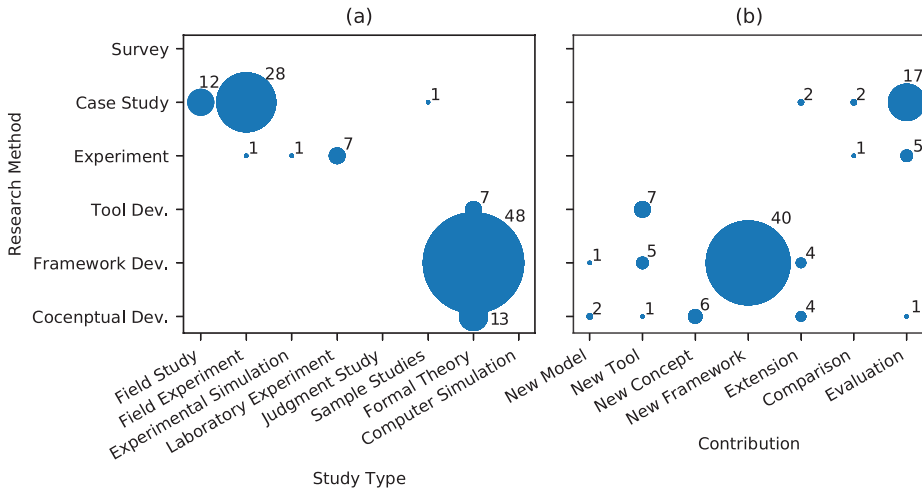


Figure 2.4: Papers mapping between a) research method and study type b) research method and contribution

In Figure 2.4 the y-axis represents the research method while the x-axis in Figure 2.4 (a) represents the study type and in Figure 2.4 (b) the contribution of the paper. The study type is based on the classification in Section 3.1.

In Figure 2.5, the y-axis represents the performance aspect while the x-axis represents the application type (grouped). We grouped the applications based on the category, purpose, and platform, e.g., web application, mobile, and embedded system.

Figure 2.6 represents the number of publications related to the topic model-based performance testing. The Figures 2.3, 2.4, 2.5 and 2.6 are based on the results of Dias Neto et al. [9] (for the period 1990-2009) and our research (for the period 2009-2019). We combined the results from the two mentioned studies and present the combined results in these figures.

4.2 Performance requirements in SRS documents

The SRS collection contained 77 SRS documents; 34 documents contained at least one performance requirement, and 43 documents specified no performance require-

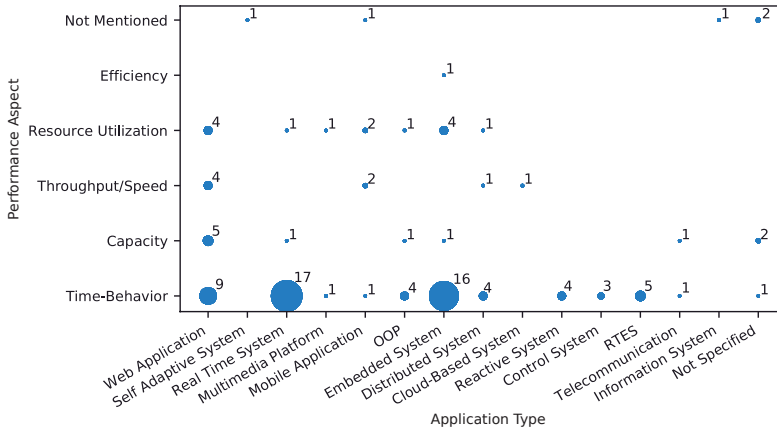


Figure 2.5: Papers mapping between performance aspect and application type

ments.

Figure 2.7 shows the mapping of the extracted performance requirements from the SRS collection. The mapping has two dimensions, representing the performance aspect that the requirement belongs to and the application type specified in the SRS document.

To extract the performance requirements we applied the coding described in Section 3.2. The total number of quantifiable performance requirements was 149. However, only 106 requirements were actually quantified, thus could be modeled and tested. Figure 2.8 shows the number of extracted performance requirements per performance aspect and the quantified requirements per aspect.

4.3 Discussion

Research on model-based performance testing has gained momentum over the past 30 years (Figure 2.6).

Performance aspects were studied to a different extent. By far the most prevalent performance aspect in studies in the context of MBT is time behavior with 66 instances³ in terms of both testing level and model used (Figure 2.3). Resource utilization, capacity, and speed/throughput were in close range with a median value of 10 instances in terms of both testing level and model used. Efficiency was the least studied performance aspect in the context of MBT, where it only appeared in one instance in terms of testing level and one in terms of the model used.

We observe a similar trend analysing the requirements specifications. Time-behavior was the most common performance aspect (Figure 2.7). Out of the 149

³We mean by instance how many times it appeared per category rather than per paper

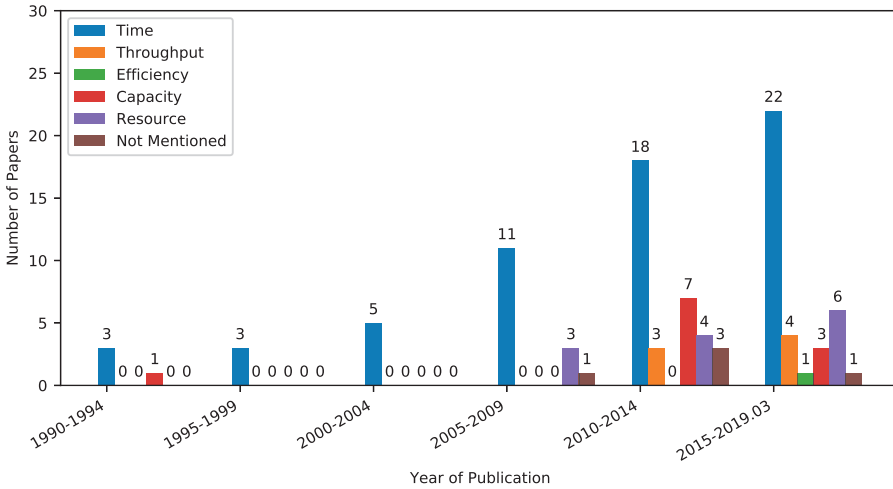


Figure 2.6: Number of publications between 1990 and 2019 - 03

extracted performance requirements, time behavior was specified in (71) requirements (e.g., *The system shall be able to search for a specified product in less than 1 second.*⁴), followed by capacity (38) (e.g., *The system must handle at least 100 concurrent users and their operations*⁵), speed/throughput (18) (e.g., *The system shall be able to retrieve 200 products per second.*⁶), efficiency (13) (e.g., *Management – all management software functions shall take optimal advantage of all language, compiler and system features and resources to reduce overheads to the minimum practical level.*⁷) and resource utilization (9) (e.g., *The FTSS software and the Vx-Works operating system, together shall [SRS193] utilize no more than 3 megabytes of ROM.*⁸).

We can see from Figure 2.7 that most of the SRS documents with performance requirements were written for web applications, followed by real-time and embedded systems. There was a diversity in terms of performance aspects in the specified requirements for web applications, whereas for real-time systems and embedded systems the specified requirements were mostly related to time behavior. A similar observation can be made by looking at Figure 2.5 where web application and embedded system appeared in 22 instances each and real-time systems in 19

⁴0000-gamma (the id of the SRS)

⁵2008-fiber

⁶0000-gamma

⁷2002-evla back

⁸2000-nasa

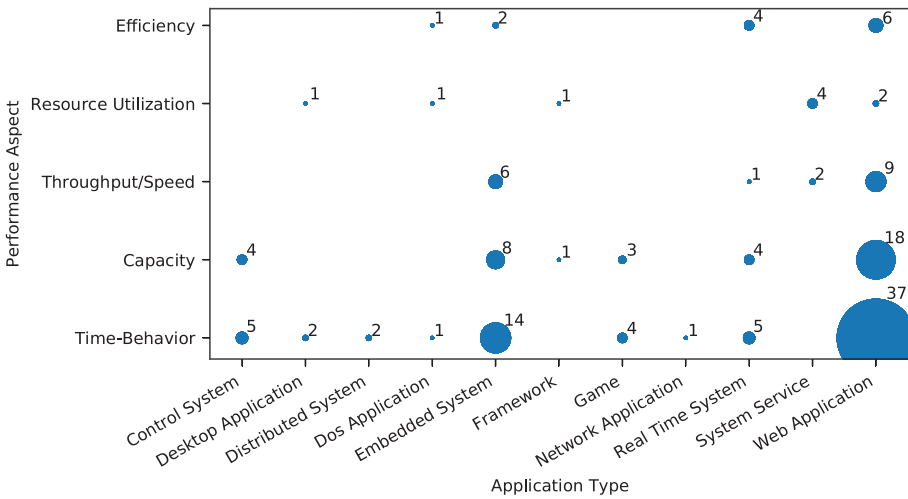


Figure 2.7: Mapping of extracted requirements between performance aspect and application type

instances. The importance of performance in web application, embedded systems and real-time system is not surprising. In a web application a large number of application users are distributed and use different communication media to access the application. Embedded and real-time systems are crucial to perform optimally, since a safety hazard could arise if performance is not addressed. For instance, in self-driving cars the time behavior for reading the value of a sensor is crucial and needs to be specified explicitly, allowing the product to be tested against that specification.

In both the identified primary studies and the reviewed SRSs, time behavior was the most common performance aspect. Nonetheless, the other performance aspects are also relevant, since they appeared in a median of 10 instances each (except efficiency) and specified in 78 requirements combined. That said, we should consider all the performance aspects when modeling performance requirements. Efficiency was the least studied (found in one paper [97]), and the least quantified in (3) requirements (Figure 2.8). However, efficiency was specified in (13) requirements, from which we conclude that efficiency is difficult to document and quantify. We found few examples of quantified efficiency requirements: 1) *The external server data store containing RLCS status for use by external systems shall be updated once per minute*⁹, and 2) *The system must accomplish 90% for transactions in less than*

⁹2004-rlcs

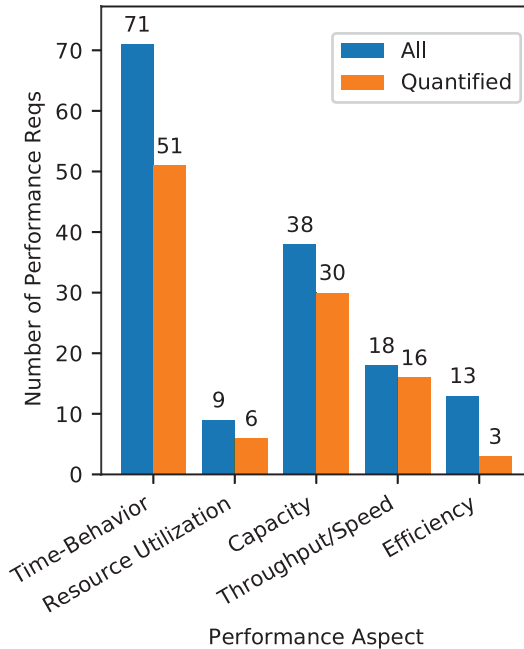


Figure 2.8: The frequency of total and quantified performance requirements per performance aspect

1 second.¹⁰ The examples show that it is possible to quantify efficiency. In the first requirement "only once every minute" and in the second "90%...less than 1 second". Both combine two performance aspects, i.e., capacity and time behavior.

Looking at testing levels, performance testing research seems to focus on system level testing (Figure 2.3). This observation coincides with the notion that software performance is not associated with a single function, but rather associated with the overall system and influenced by its structure. This is also shown in the performance requirements models used in MBT. The purpose of those models is to verify the overall system behavior, e.g., timed-automata [98, 93, 99] and behavior models [84, 100].

We extracted 50 performance requirements models and categorized them into 11 main categories (Figure 2.3)¹¹. All 11 categories had models which were used to model time behavior requirements. The purpose of those models is to verify if the written requirements are met. This is accomplished by comparing the testing results with the corresponding performance requirements.

¹⁰2008-viper

¹¹The clustering of those 50 models into 11 clusters is available online [96].

The most studied models were timed-automata and UML-related diagrams. Timed-automata were used to model and analyze the time behavior by measuring time differences between different states, which can model and verify time behavior aspects of software performance. However, timed-automata models have two main drawbacks. First, the models do not make the factors influencing performance explicit, which is needed to generate better test cases for performance requirements. Second, timed-automata can only model time behavior and are unable to cover other performance aspects [101], and are therefore only adequate when time behavior is the only performance aspect that needs to be tested. As we can see from our analysis of SRSs, time behavior is seldom the only performance requirement. UML-based models use an annotation approach to make the performance requirements more intuitive and the system behavior more understandable [84]. UML-based models solely document performance requirements, and are not used for test case generation of performance requirements. In many cases where UML is used, the performance requirements (e.g., time behavior, or capacity) is set on the model as annotation, which is later used during the test generation to add an extra assert to check this requirement. This model annotation is beneficial to verify the performance constraints of a functional requirement in a test-environment (machine resources and test data).

The models and frameworks that we extracted during the SMS were mostly newly developed with little to no validation [102, 103, 104, 105] as seen in Figure 2.4 (a). Although 31 case studies exist that validate those models (e.g., timed-automata), researchers still develop new performance requirements models and testing frameworks (Figure 2.4 (b)). The reasons for developing those models and frameworks are various:

1. Model-based performance testing in a specific field has not been done before, e.g., robotics [84], self-adaptive systems [106] and cloud API [107], has not been studied for a specific performance aspect, e.g., resource utilization [100, 108, 105] or time behavior [109], or has not been proposed in a particular development stage, e.g., early before a prototype is created [110], or late during run-time [111].
2. Issues associated with human factors where it is difficult to understand the model [112, 113], it takes extra effort to create the model [94, 114], or the current approaches are prone to human error [115].
3. The lack of automation in the current MBT approaches [116, 117, 118, 119]
4. Others reasons, e.g., using petri nets to model time behavior aspects [120].

A majority of the analyzed papers (46) suggest a new concept or framework for MBT, using formal theory research (Figure 2.4). This set is followed by 41 papers conducting field studies and field experiments that aim at validating the new model

presented in the same paper. This focus on theoretical work and studies in a relative controlled environment is another indication that the models are not validated under realistic conditions, as also observed by Prenninger et al. in their review of eight case studies on MBT [121]. A similar observation can be made by looking at the contribution of these papers in Figure 2.4 (b) where most papers introduced new ideas and methods rather than evaluating pre-existing models. It would be crucial to evaluate those models, as the lack of evaluation of MBT techniques poses a risk factor of using those techniques in industry practice. This factor influences the techniques' reliability, and evaluated techniques would positively affect their adoption in future software projects [9].

4.4 Implications of the SMS on performance requirements in MBT

We gained useful insights on performance requirements modeling in the context of MBT by conducting the SMS. First, performance requirements that were not studied before, (e.g., resources and speed/throughput), gained interest in recent years, as seen in Figure 2.6. This is an indication that more research is required in these aspects. Second, some performance attributes (e.g., time behavior) were used as test verdicts [115], while others (e.g., capacity) were used as a foundation to the test environments [122]. Third, performance requirements could be modeled separately from functional requirements, and test environments could be generated from the model [123].

However, we argue that the performance requirements models found by our SMS (Figure 2.3), do not satisfy all goals of MBT simultaneously, i.e., support requirements validation, ensure requirements testability, and support test case generation. Therefore, we developed PRO-TEST to aid the model-based performance testing process, which we introduce next.

5 PRO-TEST

In this section, we introduce and evaluate the Performance Requirements Verification and Test EnvironmentS generation approach (PRO-TEST). PRO-TEST aims at checking the completeness and correctness of performance requirements and at generating the parameters of test environments.

Figure 2.9 illustrates the MBT process in the context of performance testing. The figure is a modified version of Utting et al.'s [10] MBT process diagram that we introduced in Section 2.2. The modified process steps are shown with dotted arrows, and the modified/added artifacts are filled in grey color. We made three modifications to the diagram. First, we split the step of requirements modeling into two sub-steps: functional modeling (1-1) where a model is created from the functional requirements, and performance requirements modeling (1-2) where

performance requirements are modeled. Second, we added an iterative process between the requirements and the created models (functional and performance). This change underlines how MBT supports requirements validation (an MBT goal). The modeling stage should detect requirements issues and changes should be made to the requirements to fix these issues. Third, we added a new *Step 5*, in which test environments are generated from the performance requirements model. The software performance is thereby directly related to the test environment. Setting up a test environment requires specifying setup parameters (e.g., capacity of users) and metrics parameters (e.g., response time). These parameters are derived from the performance requirements models.

In summary, PRO-TEST consists of 1) performance requirements modeling and 2) test environment generation. These activities correspond respectively to step 1-2 and step 5 in Figure 2.9. The approach is not meant to be used as standalone but rather accompanied by any MBT approach that generates functional test cases, which results in functional test cases mapped to test environments that test the performance of the SUT.

We illustrate PRO-TEST's core concepts in Section 5.1 and explain the steps and guidelines for creating the performance requirements model in Section 5.2. Additionally, we explain the steps of generating test environments in Section 5.3, illustrate its application on an example in Section 5.4, and apply PRO-TEST on a set of 149 performance requirements in Section 5.5, discussing the strengths and weaknesses of the approach.

5.1 PRO-TEST approach development and description

We intend to propose a modeling approach that addresses the limitations of current model-based performance testing. Specifically, the modeling of the five performance aspects in Section 2.1 to verify the requirements while generating test environments. Looking at the existing approaches that we identified in our SMS, we found that performance requirements affect test environments. Therefore, instead of creating an approach that models both performance and functional requirements, we chose to develop an approach that focuses on modeling performance requirements and generating test environments. This approach can be accompanied by existing well-established MBT approaches that already handle functional modeling and testing. By focusing on performance requirements, we increase the chance of our approach being used by practitioners who are already using existing MBT for functional testing, without the need to replace their existing tools but add to what they already use.

The development of PRO-TEST was inspired by two related principles. First, the experiment principle that illustrates the relationship between dependent and independent variables [124]. Second, cause-effect graphs (CEGs) [125] that can be used to model the relationships between causes and effects.

We analyzed the different performance aspects while having the cause-effect

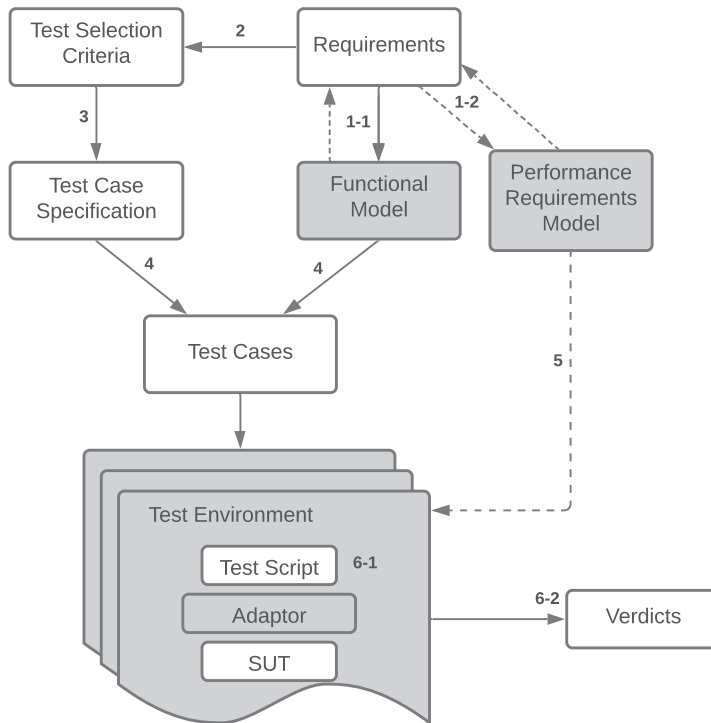


Figure 2.9: MBT Process in the context of performance testing

concept in mind. The main insight we had is that one set of performance requirements (capacity, resource constraints) can influence another set (time behavior, throughput, efficiency). This concept is shown in a taxonomy tree (Figure 2.10) that classifies the aspects in independent and dependent performance parameters. The independent parameters consist of capacity (e.g., the maximum number of users), and resource constraint (e.g., storage size), which represent constraints on the software. The dependent parameters consist of time behavior (e.g., response time), throughput/speed (e.g., requests per time unit), and efficiency (e.g., response time in regards to memory size), and are measurements of the software performance. The manipulation of the independent parameters causes changes in the dependent parameters. For example, if we require the system to use fewer resources (all other things being equal), it would lead to a higher response time, lower throughput, or efficiency. The purpose of this taxonomy tree is to identify which performance requirements are the influencing factors and which ones are impacted,

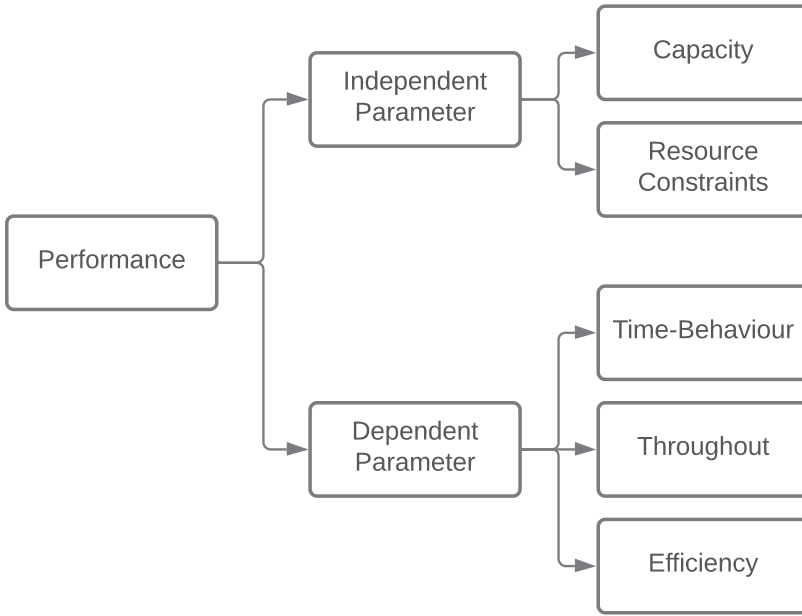


Figure 2.10: Performance Parameters Taxonomy

as this is important to distinguish when modeling testable system requirements. The taxonomy tree is by no means exhaustive, but rather a classification of the most common (studied and specified) performance requirements.

In the previous paragraph, we used the term resource "constraints" instead of "utilization" in order to emphasize the interpretation of the parameters as an independent parameter. Looking at our results from the SRS analysis, we found that the specified resource utilization requirements could be both a dependent variable that we measure when we run the tests or an independent variable that affects the dependent variables when constructing and running the tests. For example, if we take the requirement *"The FTSS software and the VxWorks operating system, together shall [SRS193] utilize no more than 3 megabytes of ROM."*¹², there are two methods to test it. Firstly, we run the tests, measure the utilized ROM, and make sure the software does not utilize more than 3 megabytes. Alternatively, we set up a test environment with 3 megabytes of ROM as a constraint, run the tests, and if the tests run completely, then the software satisfies this requirement. We chose to apply the second method (hence the use of terminology resource constraints) since

¹²2000-nasa

it works better when the specified requirement affects our decision when setting up the test environment. For instance, to test the requirement "GParted is not a resource hog and will run on almost every computer"¹³, we can't run the tests and measure the utilized resources (even if this requirement is to be quantified). Instead we need to define a set of representative computers and run the tests on them.



Figure 2.11: Performance requirements model

Figure 2.11 presents the main components of a performance requirements model. The model consists of three main parts:

1. The object element referring to the SUT or part of it, i.e., a function that has the performance requirements associated with it.
2. The independent parameters which act as inputs. They affect the test environment where the test runs and affect the test data.
3. The dependent parameters which act as outputs. They are the metrics or results of running the tests, used to compare the test results with the written performance requirements.

Performance requirements are modeled with PRO-TEST using the taxonomy tree that acts as a guide when extracting, categorizing, and finding missing performance requirements.

5.2 Performance requirements model

There are three steps that should be followed when modeling the performance requirements of the software.

- *Step 1: Define the objects.* Look up the object that the performance requirements on hand applies to. The objects could be the system, specific functions, or a collection of functions.
- *Step 2: Define the independent and dependent parameters.* Extract the performance parameters from the requirements, and code them with the appropriate performance aspect using the taxonomy tree. Then add those parameters to the corresponding model.

¹³2010-gparted

- *Step 3: Compare the model with the taxonomy tree.* Take the created performance requirements model and compare it with the taxonomy tree. Look for any possible missing parameters. If some parameters are missing, look for the possibility of merging models with the same object. If there are still some missing parameters, then there is a problem with the requirements. Check with requirements engineers or customers to negotiate the requirements. Otherwise, the model is complete and the specified requirements are quantified and can be tested. When the modeling is done, the next step is to design the test suite.

When using PRO-TEST with performance requirements, one should take into consideration the following guidelines which help to model the requirements.

- *Guideline 1: Verify the completeness of the requirements.* Check the relation between different requirements. There should be a correspondent independent input for each dependent output. Having one without the other would result in ambiguous requirements, which would reflect an incomplete performance requirements model.
- *Guideline 2: Verify feasibility.* The requirement should fit with one of the performance aspects' definitions in Section 2.1.
- *Guideline 3: Verify quantifiability.* Each requirement should have a quantity that describes the target level of performance, and an object that specifies where the target level applies (system, a specific function, or a collection of functions).
- *Guideline 4: Specific condition.* Check if the requirements apply in specific circumstances or scenarios. The performance requirements might have the same objects but under different conditions, i.e., peak time. In this case, one should make a different model for each of those conditions, because each condition has different parameters that apply to the test environment and different measurement levels.
- *Guideline 5: Mandatory performance aspects.* To generate meaningful test environments, each model requires the following performance aspects to be specified: 1) capacity and resource constraints to help set up the test environment, and 2) time behavior or throughput which acts as the metric to measure when running a test.

While these suggestions stem from our experience of modeling nearly 150 performance requirements from 34 SRS documents, they are not exhaustive and should not be considered as rules.

5.3 Generating Test Environments

One of the goals of PRO-TEST is to generate test environments, which aids the verification of performance requirements in the SUT. As seen in Figure 2.9, the generated test environments are required to run performance test and affect the outcome of performance tests.

Using the created performance requirements models, we generate parameters for the test environments. These parameters are divided into two groups: *constraints* and *metrics*. The *constraints* parameters are required to set up the test environments and stem from the independent parameters in the taxonomy in Figure 2.10. The *metrics* parameters are indicators for the success or failure of the test cases run in the test environment, and stem from the dependent parameters in the taxonomy in Figure 2.10.

```

1 Create constraintsList
2 Create metricsList
3 Add resource constraints to constraintsList
4 Add capacity to constraintsList
5 Add time behavior to metricsList
6 Add speed/throughput to metricsList
7 Add efficiency to metricsList
8
9 Create environmentsList
10 CALL testEnvGenerator with constraintsList and metricsList
11 Add the generated environment to the environmentsList
12 FOR each constraint in constraintsList
13     CALL testEnvGenerator with constraint and metricsList
14     Add the generated environment to environmentsList
15 END FOR
16
17 CALL mapTestCasesToEnvironments with environmentsList

```

Listing 2.1: Test Environment Generation Algorithm

We show in Listing 2.1 the algorithm to generate the test environments that will be used to run the test cases. The algorithm consists of three main steps. 1) Create two lists of parameters, *constraintsList* and *metricsList*, and add the parameters from the created performance requirements models to the corresponding list based on the classification in the taxonomy tree. 2) Create an *environmentsList*, one for each parameter in the *constraintsList* with all parameters in the *metricsList*, and an environment where all parameters in *constraintsList* and *metricsList* are included. 3) Map the test cases to the created environments in *environmentsList*. The test cases mapped to the test environments are those that verify the object (e.g., function) to which the performance requirements refer.

To automatically generate test environments from the created performance requirements model, we implemented a Python script ¹⁴. The script takes as input the list of performance requirements models (in CSV format) created by the tester.

¹⁴The test generation script is available online [96]

The output of the script is a list of test environments (in JSON format). Each test environment consists of a list of constraints to construct the test environment and object-metric pairs that indicate what functions should be tested and measured in this environment. We chose JSON as output format since it is a widely used in practice. Generating test environments in this format makes it fairly easy to adapt to different testing tools.

5.4 Example of PRO-TEST

To illustrate PRO-TEST, we present an example, following the three steps described in Section 5.2 for creating the performance requirements model. Then, we generate parameters for test environments following the test environment generation presented in Section 5.3. We extracted performance requirements for a telescope control software shown in Table 2.4.

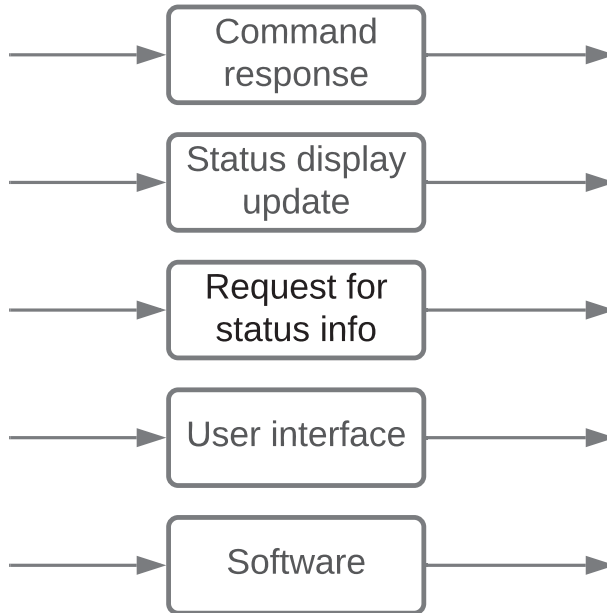


Figure 2.12: PRO-TEST Example — Step 1

Table 2.4: Example Performance Requirements for PRO-TEST Approach Demonstration

No.	Performance Requirements	Performance Aspect
PR1	The Gemini <u>software</u> should have no hard restrictions on the number of simultaneous users , but should allow for policy decisions that do restrict the amount of simultaneous access.	Capacity
PR2	Every command must be accepted/rejected within 2 sec and before the corresponding action occurs. (This is different than the ACK-/NAK response of the communications protocol - here, the target system must have examined the command and verified its validity.)	Time Behavior
PR3	Status display update must be within 4 sec at the local stations (certain functions, such as telescope position, may have tighter constraints). Remote station update response is given in the Requirements for Remote Operations section.	Time Behavior
PR4	Requests of subsystems for status information must be answered within 5 sec and be possible in maintenance level operation.	Time Behavior
PR5	Requirements for response times within the <u>user interfaces</u> are given in the User Interface requirements section.	Time Behavior
PR6	The <u>user interface</u> should rather be seen as a package to be callable from a large number of stations , depending on where a user is.	Capacity
PR7	The <u>user interface</u> should also be network transparent so that it does not matter where it is being run.	Resource Constraints
PR8	As a conclusion, the Gemini 8m Telescopes control <u>software</u> shall allow simultaneous operation of up to six active control nodes and up to two more monitoring nodes (one local and one remote) without appreciable degradation of performance.	Capacity
PR9	In practice the operation and facilities foreseen so far for the Gemini 8m Telescopes will limit this number to a maximum in the order of three active nodes, but the Gemini 8m Telescopes computers and <u>software</u> shall be capable of coping with the load of 10 active nodes , should the case arise.	Capacity
PR10	All software bugs should be logged and then fixed as soon as possible after detection. The goal is to have restart conditions occur only on hardware failure. Fault recovery, exception handling, fail-safe checks, etc. should be used to improve reliability.	Availability

The requirements in this table were extracted from the SRS document 1995-gemini

Performance requirements model

Step 1: Define the objects. We defined five objects from the requirements: command response, status display update, request for status info, user interface and software. Then we created five models, one for each object as shown in Figure 2.12.

Step 2: Define the independent and dependent parameters. We extracted the performance parameters (10 active nodes, large number of stations, simultaneous users, 6 active control nodes & 2 monitoring nodes, ≤ 2 sec, ≤ 4 sec, ≤ 5 sec and network) from the requirements, and coded them with the related performance aspects as per the taxonomy tree. We present the associated performance aspect in the last column of Table 2.4. Then we added those parameters as independent and dependent parameters in the model as shown in Figure 2.13. At this stage we

identified four issues in the requirements:

1. PR10 is an availability requirement, which is not to be found in our taxonomy tree (guideline 2), hence we exclude PR10. 2) PR5 indicates that there should be a time behavior requirement for the user interface. However, we examined the SRS document and we did not find any time behavior requirements for the user interface. Hence, PR5 can not be modeled and it indicates a missing requirement.
2. PR1 (simultaneous users), PR6 (large number of stations), and PR7 (network) are not quantified (guideline 3).
3. PR6 is ambiguous as "without appreciable degradation of performance" is not unclear.
4. PR8 and PR9 are conflicting requirements. PR8 specifies a capacity of 8 nodes (6 active plus 2 monitoring), however, PR9 specifies a capacity of 10 active nodes.

Step 3: Compare the model with the taxonomy tree. We compared the created model with the taxonomy tree to identify any possible missing parameters. We put the possible missing requirements on each corresponding model as seen in Figure 2.14. Resource constraints parameters are missing from the models and the specified requirements for the software since there were no requirements indicating resource constraints. Another issue is that the requirement PR5 (large number of stations) applies to other parts of the system as well (missing requirement). Moreover, there are no performance requirements from the dependent parameters (time behavior, speed/throughput, or efficiency) that apply to the software or the user interface.

At this point of the analysis, the identified issues should be discussed with the requirements engineers or customers to negotiate the requirements and fix the issues: asking for 1) the missing requirements, 2) quantify PR1, PR6, and PR7, 3) clarify or reformulate the existing requirement PR8 into two requirements, one that specifies the capacity for the software, and the other that specifies the dependent parameter e.g., time behavior, and 4) resolve the conflict in the requirements PR8 and PR9.

Test environments generation

We generate test environments parameters following the test environment generation algorithm presented in Listing 2.1. We feed each model in Figure 2.14 (constraints and metrics) to the algorithm as input, and as output we get a set of environments (one per constraint and one with all constraints). This makes debugging easier, as the tester can identify the troublesome performance constraint(s)

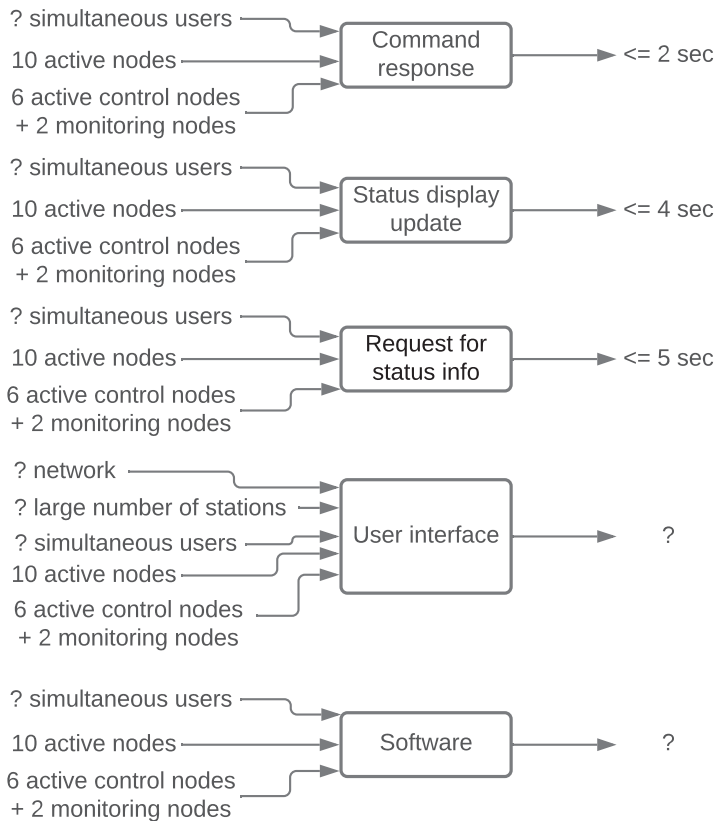


Figure 2.13: PRO-TEST Example — Step 2

just by looking at the constraint(s) used to construct the test environments in which the failed test was run.

We used our test environment generations script to automatically generate test environments from the created models. In Figure 2.15, we show the structure of the generated file. The root node of the file contains an array of generated test environments. Each test environment consists of a list of *constraints* and a list of *object-metric pairs*. A *constraint* presented using a *description* and an *att class* (the performance aspect). An *object-metric pair* consists of the object to be tested (e.g., a function), and the metric to be measured. A metric is presented using a *description* and *att class*. Errors in the modeled performance requirements will be shown in *errors*.

The results of generating test environments can be found in Table 2.5. The

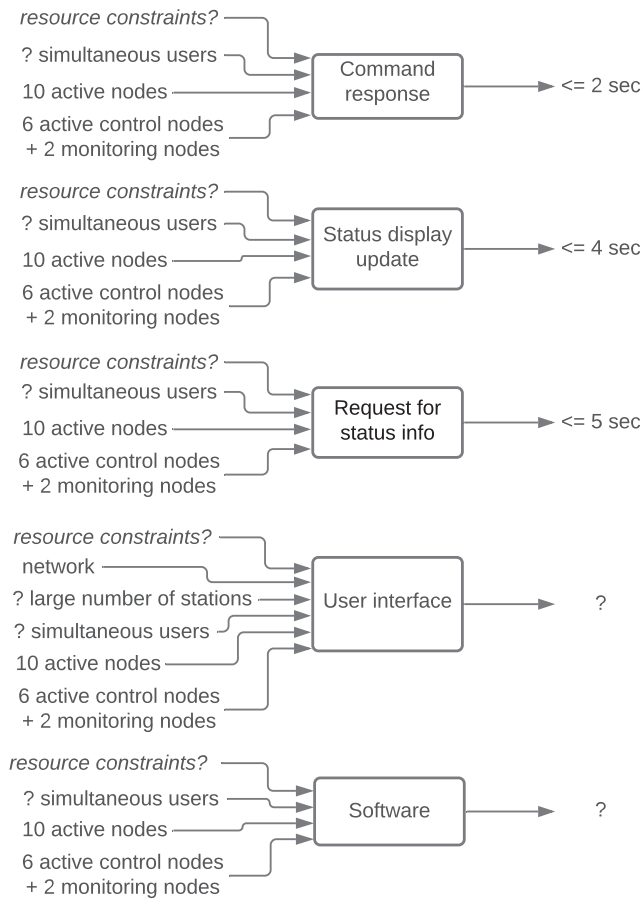


Figure 2.14: PRO-TEST Example — Step 3

rows 1-4 can be used to construct test environments. This is not possible for the remaining rows (5-8), as they are missing constraints and/or metrics. For instance, the question mark in row 5 for the constraint *simultaneous users* is an indication of a missing quantity of *simultaneous users*. As we mentioned earlier in this section the requirements should be negotiated with the customer, so we can fill the gaps in our tests.

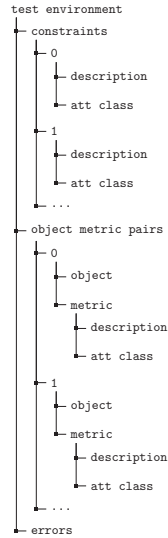


Figure 2.15: Test environment JSON file structure

5.5 Sample study - model evaluation

We applied PRO-TEST on 34 SRS documents from the SRS collection. We extracted in total 149 performance requirements from the SRS documents, i.e., requirements that fit the definition of performance aspects in Section 2.1.

We extracted the performance requirements from the SRS collection and applied PRO-TEST by modeling the requirements as explained in Section 5.2. We did not generate test environments from the created performance relational models, since test environments generation would be more meaningful if used with another MBT approach to generate test cases from functional requirements. This is outside the scope of this paper.

In Table 2.6 we present two types of defects found by PRO-TEST. The first defect is related to quantifiability. We found that 106 out of 149 requirements were quantified, while the remaining 43 were quantifiable but were not actually quantified (e.g., *"The product will reside on the Internet so more than one user can access the product and download its content for use on their computer."*¹⁵).

The second type of defect is related to under-specified or missing requirements. We found a total of 180 missing parameters in the analyzed requirements. The majority of them (100) were related to resource constraints, followed by capacity (39), time behavior (22), and throughput/speed (19). No missing parameters for efficiency requirements were detected. As defined in Section 2.1, efficiency is a

¹⁵2001-space fraction

Table 2.5: PRO-TEST Example - Test Environments Summary

Id	Constraints	Object (Measure)
1	10 nodes	Command Response (≤ 2 sec), status display update (≤ 4 sec), request for status info (≤ 5 sec), software
2	100 simultaneous users	Command Response (≤ 2 sec), status display update (≤ 4 sec), request for status info (≤ 5 sec), software
3	10 nodes, 100 simultaneous users	Command Response (≤ 2 sec), status display update (≤ 4 sec), request for status info (≤ 5 sec), software
4	10 nodes	User interface
5	? simultaneous users	User interface
6	? network	User interface
7	? number of stations	User interface
8	10 nodes, ? simultaneous users, ? network, ? number of stations	User interface

Table 2.6: PRO-TEST evaluation results

Defect	Quantity
Not-quantified Requirements	43
Under-specified Parameters	180
Under-specified Resource constraints	100
Under-specified Capacity	39
Under-specified Time-behavior	22
Under-specified throughput	19
Under-specified Efficiency	0

combination of more than one parameter. Hence, to some extent, the existence of those parameters (e.g., time behavior and resources constraints) eliminates the need for efficiency requirements.

In the included SRS documents there were 204 performance requirements, categorized by the original author of the SRS; We identified and categorized 132 of those requirements, while we could not fit 67 requirements to any of the performance aspects definitions in Section 2.1. For example, *"Assuming submitted statistics for jobs are accurate, the Libra scheduler will ensure that all jobs are completed with a 10% error allowance."*¹⁶. Other requirements were hard to understand how they fit in performance requirements, e.g., *"The database retrieval and update response time shall not impact any other performance requirements such as the GUI response time or monitoring and control responses."*¹⁷. This requirement mentions response time, but it does not clearly state where does it apply or what the target level of performance is. There were some requirements that were more difficult to identify,

¹⁶2001-libra

¹⁷2004-rlcs

e.g., *"The HATS-GUI shall allow a user to request transformations while HATS-SML is performing transformations or parsing."*¹⁸. It could be argued that this requirement is an efficiency requirement. But reading it carefully we concluded that this is not a performance requirement, but rather a usability requirement that demands parallel processing or multitasking. According to Ho et al. [126], a performance requirement can be categorized into four levels (0 to 3). These levels show the maturity, suitability, and validation of performance requirements. Based on their definition, this requirement is classified as level 0 (lowest), which is descriptive and can only be evaluated qualitatively. The requirements in this paragraph were extracted from 2001-libra, an SRS for economy-driven cluster scheduler for high-performance clusters, 2004-rlcs, an SRS for an interstate reversible lane control system, and 2001-hats, a high assurance transformation system. Relying solely on a qualitative evaluation of performance in these systems leads potentially to unsatisfied customers.

Out of the 149 requirements, 43 were not quantified. Those requirements fall into two categories. (1) Requirements with minor issues, i.e., just missing the numerical value. For example *"The tools shall be able to scale to process large collections using distributed processing and data transport."*¹⁹. This is a capacity requirement, that applies to the whole tool (object). However, the size of the collection is not defined; it could be 100 or 100,000. Since the requirement does not specify a range, we do not know how to test it. (2) Requirements with major issues. For example *"Loading speed: The data system shall load as quickly as comparable productivity tools on whatever environment it is running in."*²⁰. This requirement refers to efficiency in an ambiguous manner: "as quick as possible" and "on whatever environment". No test could be written to verify if the system satisfies this requirement.

Performance aspects were not considered equally by the requirements engineers when writing the SRS documents, which shows the lack of knowledge in the interdependency relation between different aspects as shown by PRO-TEST. 100 out of 109 created models had missing requirements in resource constraints. It could be argued that resource constraints are not a part of performance requirements. However, it does affect software performance, and there were some SRS documents that specified resource constraints properly e.g., *"The Framework Shell SHOULD NOT utilize more than 40 megabytes of RAM."*²¹.

We generated 96 test environments from the performance requirements models that we created from the SRS collection. All of the generated test environments had missing or unquantified requirements.

Bondi [127] suggested that a performance requirement should have nine characteristics: unambiguous, measurable, verifiable, complete, correct, mathematically

¹⁸2001-hats

¹⁹2009- warc III

²⁰2006-stewards

²¹2005-znix

consistent, testable, traceable, and can be linked to business and engineering needs. Our study corroborates that PRO-TEST supports a subset of these characteristics: it helps engineers in verifying performance characteristics as it makes lack of information explicit (completeness and quantifiability), it detects unclear information (ambiguity), and associates performance requirements to test environments.

6 Discussion

In this section, we discuss the different aspects of PRO-TEST. We compare the performance taxonomy and the performance aspect inter-dependency relation with those from the literature, list the limitations of the approach, discuss how the approach differs from other MBT approaches, show our observations regarding performance requirements, and finally we discuss PRO-TEST with performance prediction.

6.1 Previous performance aspect classifications

As we saw from our SMS and SRS analysis results, five performance aspects were studied and used in practice. Thus, testers should consider these aspects when testing software performance. Eckhardt et al. [128] specify a template to write performance requirements. They considered three aspects of performance requirements, namely time behavior, throughput and capacity. In addition, they specified performance context (e.g., platform, measurement location and load) as part of each requirement. However, they do not consider resource constraints, but rather the platform (hardware) under which the requirement applies. It is seldom the case that specifying hardware requirements is enough to test system performance and ensure the desired time behavior, throughput and efficiency. For instance, smartphone applications, vehicles software, cloud services, and desktop applications, all share resources with other applications running on the same platform. In this case, performance testing verdicts are more reliable when we specify the available resources for the system rather than the platform it runs on. Nixon et al. [129] categorized performance requirements into time (response time, throughput and management time) and space (main memory, secondary storage). They did not account for capacity which we consider in our taxonomy tree.

6.2 Performance aspects inter-dependency

The dependency relation between the five performance aspects as far as we know was not observed before. Cai et al. [130] considered two aspects of performance, time and space, and called the relation between these aspects side-effects. They did not define clearly the nature of the effect, nor considered the other performance aspects. Eckhardt et al. [128] proposed that each specified performance requirement

should have platform and load in the same requirement, since these aspects affect all other performance aspects. They do not consider the case when platform and load requirements are specified in separate requirements, which can be the case as we saw in our SRS analysis results.

6.3 PRO-TEST benefits

Using PRO-TEST to model performance requirements and generate test environments has the following benefits:

1. It helps software engineers to understand the requirements better. When the performance requirements are visualized and by using the taxonomy tree, it becomes easier to find the relation between the requirements and how they relate to functional requirements
2. It acts as a validation tool for the requirements. By modeling the performance requirements, we can find out 1) if there are issues with some requirements, which can not be modeled, and 2) if other requirements are missing.
3. It informs software testers in what environments the tests should be run. This saves time and resources as it allows testers design efficient test suites.

6.4 PRO-TEST limitations

There are some limitation of using this modeling approach to model performance requirements. First, the taxonomy tree is rather abstract. By using the taxonomy, we can identify that capacity requirements are missing, however, currently it provides no support or details about what is missing, e.g., data, users, requests. These could be specified in more detail in further nodes of the taxonomy. Second, the approach is prone to human error. Since the extraction and coding of the parameters is done manually, the process depends on the engineers' interpretation of the requirements. This could be avoided by automating the process using natural language processing. Fourth, a lack of inspection of the requirements' quality. As argued by Bondi [127] a good performance requirement should specify to what degree a requirement should be met, i.e., we should specify if the requirement applies all the time or a specific amount of the time (99% of the time). Using the PRO-TEST we do not detect those quality aspects of the requirements.

The main limitation of our approach of test environments generation, is that it can be difficult for a tester to debug the failed performance test. PRO-TEST generates one test environment per constraint, in addition to a test environment that aggregates all constraints. If performance tests fail in the test environment that aggregates all the constraints, then it is difficult to identify which interaction of the test constrains is the cause of the failure.

6.5 Observations on dependent and independent parameters

The dependent parameters (time behavior, throughput, efficiency) were more often specified than independent parameters (capacity, resource constraints) in performance requirements. This is clear from the results, where out of the 180 under-specified requirements 139 missing requirements were under the category of independent parameters (i.e., capacity and resource constraints). There could be many reasons for this outcome. First, it is possible that some requirements engineers or customers have a misconception when it comes to some performance aspects. Resource constraints could be thought of as part of hardware specifications. Second, it may be more difficult to specify those parameters during the initial stage of a software development cycle. If no prior experience exists it is difficult to assess how much resources are utilized or capacity required, i.e., no clear estimation existed about capacity. This increases the risk of scalability issues appearing later. Similar to what happened at the PokemonGo launch [68], as the developers did not expect the big surge in the number of users. Third, resource constraints was left out intentionally. Today hardware virtualization is used extensively in deployed applications, it is very flexible and affordable to invest in higher specs hardware than more efficient software.

6.6 Performance prediction

Performance prediction is an approach to ensure the performance of the system by simulating the system behavior. Similar to MBT, performance prediction can use models to illustrate the system behavior [131, 69]. Performance prediction is used to validate the system performance early before building the system (e.g., in a simulated environment) [131]. In contrast, PRO-TEST verifies performance requirements through modeling and generates test environments for performance testing.

Performance prediction is useful in systems with hardware components, where we want to understand the effect of the components used on the system performance. At the same time, the PRO-TEST and model-based performance testing approaches are appropriate to generate means of testing the software before deployment.

7 Answering the research questions

We answer now our four main research questions.

RQ1 *Which aspects of performance requirements are used in MBT?*

All performance aspects presented in Section 2.1 were used in MBT but to different extents. Time behavior was the most studied by researchers and specified by practitioners in the SRSs. Capacity, throughput, and resource constraints were

studied and specified but to a lesser extent compared to time behavior. Efficiency was the least studied aspect with one paper and was only quantified in about 3 out of the 13 written efficiency requirements. We found many models that can be used to model those aspects. We can see in Figure 2.3, many of the models were used to model more than one performance aspect.

RQ2 *How to implement MBT on performance requirements aspects?*

We found 50 models in the literature to model software performance requirements, and grouped them into 11 clusters (Figure 2.3). The purpose of those models is to document and visualize performance requirements. Those models do not satisfy the goals of MBT, which are 1) validate the specified requirements, 2) better understand those requirements, and 3) generate a suitable test suite. Hence, we developed PRO-TEST that consists of a model and a taxonomy tree for performance aspects, which verifies performance requirements and generate test environments. The performance requirements model with the taxonomy tree is not just a modeling approach for performance requirements. It is also a concept that identifies the relationship between different performance aspects.

RQ3 *To what extent is the identified approach effective at modeling performance requirements written for real-life projects?*

The results from PRO-TEST evaluation indicate that the developed approach can be used to model requirements from real-life projects. We applied PRO-TEST to performance requirements from 34 SRS documents. The approach could detect issues related to ambiguity, quantifiability and completeness of performance requirements. We could also understand the interrelation between those requirements. However, there are some limitations to PRO-TEST. 1) The taxonomy tree is not detailed enough, e.g., we do not know which type of capacity is missing (users, data size). 2) manually modeling the requirements is prone to human errors. Those limitations should be addressed to achieve the maximum benefits of MBT.

8 Conclusions and future work

In this study, we illustrated how PRO-TEST can improve the understanding of performance requirements and support the identification of requirement defects. We conducted a systematic mapping study in the context of model-based performance testing and studied a repository of publicly available software requirements. We found from our SMS that researchers studied and modeled all performance aspects. However, there was a need to develop an approach to verify performance requirements that takes into consideration the goals of MBT. We developed PRO-TEST and showed by our evaluative study that it can be used to verify performance requirements and generate test environments. The benefits of PRO-TEST adds value to MBT. It helps software engineers to understand the requirements better, validate them, and generate test environments semi-automatically. In addition to

the performance relational model, we developed the taxonomy tree, which shows the cause-effect relation between different performance aspects.

Future work concerns more in-depth validation of PRO-TEST, finding solutions for the limitations of the approach, extending PRO-TEST to existing diagrams, and other non-functional requirements. We have identified the following possible directions for future work, which would be of benefit to researchers who are interested in this area.

1. Apply the proposed modeling technique on a larger set of well-built SRS with relatively completed performance requirements and to enhance PRO-TEST further.
2. Investigate the possibility of implementing the relational modeling concept in other non-functional requirements, e.g., security.
3. Integrate PRO-TEST with MBT approaches that generate functional test cases, and evaluate the effectiveness of test environment generation.
4. Extend the taxonomy tree by finding the possible sub-categories for the performance aspects.
5. Automate the process of creating the model from natural language requirements to avoid human errors.

Finally, we hope that this list of future work inspires researchers to do more research in the area of model-based performance testing and performance requirements veri.

Appendix

1 Included Papers in the SMS (Chapter 2)

Table A.1: Included papers in the SMS

No.	Title	Author	Year
S1	Model-based performance testing in the cloud using the mbpet tool	Abbors et al.	2013
S2	Approaching performance testing from a model-based testing perspective	Abbors et al.	2010
S3	Model-based testing of a real-time adaptive motion planning system	Abdelgawad et al.	2017
S4	GeTeX: A Tool for Testing Real-Time Embedded Systems Using CAN Applications	AbouTrab et al.	2011
S5	Test generation for performance evaluation of mobile multimedia streaming applications	Al-tekreeti et al.	2018
S6	Dtron: a tool for distributed model-based testing of time critical applications	Anier et al.	2017
S7	Canopus: A Domain-Specific Language for Modeling Performance Testing	Bernardino et al.	2016
S8	Online model-based testing under uncertainty	Camilli et al.	2018
S9	Event-based runtime verification of temporal properties using time basic Petri nets	Camilli et al.	2017
S10	Abstracting timing information in UML state charts via temporal ordering and LOTOS	Chimisliu et al.	2011
S11	Generation of scripts for performance testing based on UML models	Da Silveira et al.	2011
S12	Timed testing under partial observability	David et al.	2009
S13	Model-Based Test Suite Generation for Function Block Diagrams Using the UPPAAL Model Checker	Enoiu et al.	2013
S14	Iterative test suites refinement for elastic computing systems	Gambi et al.	2013

APPENDIX A. APPENDIX

S15	Fast model-based test case classification for performance analysis of multimedia mp soc platforms	Gangadharan et al.	2009
S16	Fault-driven stress testing of distributed real-time software based on uml models	Garousi	2011
S17	Automated Steering of Model-Based Test Oracles to Admit Real Program Behaviors	Gay et al.	2011
S18	Model-driven testing approach for embedded systems specifics verification based on UML model transformation	Grigorjevs	2011
S19	Usage profile and platform independent automated validation of service behavior specifications	Groenda	2010
S20	A model-based testing technique for component-based real-time embedded systems	Guan et al.	2015
S21	Validating Timed Component Contracts	Guilly et al.	2015
S22	Towards effective and scalable testing for complex high-speed railway signal software	Hu et al.	2017
S23	Experiences of Applying UML/MARTE on Three Industrial Projects	Iqbal et al.	2012
S24	Environment modeling and simulation for automated testing of soft real-time embedded software	Iqbal et al.	2015
S25	Applicability of an integrated model-based testing approach for rtcs	Iyengar et al.	2011
S26	Model-Driven Method for Performance Testing	Javed et al.	2018
S27	Experience Report: Evaluating fault detection effectiveness and resource efficiency of the architecture quality assurance framework and tool	Johnsen et al.	2017
S28	Interaction-based runtime verification for systems of systems integration	Krüger et al.	2010
S29	Quality Assurance for Component-based Systems in Embedded Environments	Li et al.	2018
S30	Timed moore automata: test data generation and model checking	Löding et al.	2010
S31	Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints.	Luthmann et al.	2019
S32	Modeling and testing product lines with unbounded parametric real-time constraints	Luthmann et al.	2017
S33	Automated significant load testing for ws-bpel compositions	Maâlej et al.	2013
S34	Conformance testing for quality assurance of clustering architectures	Maâlej et al.	2013
S35	Model-based conformance testing of ws-bpel compositions	Maâlej et al.	2012
S36	Towards an industrial strength process for timed testing	Mitsching et al.	2009
S37	Comparative analysis for software testing: Mobile applications versus web applications	Muhamad et al.	2016
S38	Test Selection for Data-Flow Reactive Systems Based on Observations	Nguena-Timo et al.	2011
S39	PLeTsPerf - A Model-Based Performance Testing Tool	Rodrigues et al.	2015
S40	Evaluating capture and replay and model-based performance testing tools: an empirical comparison	Rodrigues et al.	2014
S41	Extending UML testing profile towards non-functional test modeling	Rodrigues et al.	2014
S42	An experience report on an industrial case-study about timed model-based testing with UPPAAL-TRON	Rütz et al.	2011
S43	Testing of timing properties in real-time systems: Verifying clock constraints	Saadatmand et al.	2013

S44	On Combining Model-Based Analysis and Testing	Saadatmand et al.	2013
S45	Functionality, performance, and compatibility testing: A model based approach	Saqib et al.	2018
S46	Checking response-time properties of web-service applications under stochastic user profiles	Schumi et al.	2017
S47	Analyzing a wind turbine system: From simulation to formal verification	Seceleanu et al.	2017
S48	Introduction of time and timing variability in usage model based testing	Siegl et al.	2010
S49	Partitioning the requirements of embedded systems by input/output dependency analysis for compositional creation of parallel test models	Siegl et al.	2015
S50	Multi-fragment Markov model guided online test generation for MPSoC	Vain et al.	2017
S51	Provably Correct Test Development for Timed Systems	Vain et al.	2014
S52	System Testing of Timing Requirements Based on Use Cases and Timed Automata	Wang et al.	2017
S53	A model-based framework for cloud api testing	Wang et al.	2017
S54	Towards an integrated approach for validating qualities of self-adaptive systems	Weyns	2012
S55	Vision paper: Towards model-based energy testing	Wilke et al.	2011
S56	System Modules Interaction Based Stress Testing Model	Yang et al.	2010
S57	A methodology of model-based testing for aadl flow latency in cps	Zhu et al.	2011

Table A.2: Extracted Papers from Dias-Neto 2010

No.	Title	Author	Year
S58	Specification-based testing for real-time reactive systems	Alagar et al.	2000
S59	Designing fault injection experiments using state-based model to test a space software	Ambrosio et al.	2007
S60	Generating test suites for software load testing	Avritzer et al.	1994
S61	Specification-based testing for real-time avionic systems	Biberstein et al.	1999
S62	On the correctness of upper layers of automotive systems	Botaschanjan et al.	2008
S63	Distributed software testing with specification	Chang et al.	1990
S64	Traffic-aware stress testing of distributed systems based on UML models	Garousi et al.	2006
S65	Testing from a stochastic timed system with a fault model	Hierons et al.	2009
S66	Automatic timed test case generation for Web services composition	Lallali et al.	2008
S67	Regression testing of classes based on TCOZ specification	Liang	2005
S68	Generating test cases for real-time systems from logic specifications	Mandrioli et al.	1995
S69	Derivation of tests from timed specifications according to different coverage criteria	Merayo et al.	2008
S70	T-UPPAAL: online model-based testing of real-time systems	Mikucionis et al.	2004
S71	Generating functional test cases in-the-large for time-critical systems from logic-based specifications	Morasca et al.	1996

APPENDIX A. APPENDIX

S72	Mutation-based Testing Criteria for Timeliness	Nilson et al.	2004
S73	Model-based testing in evolutionary software development	Pretschner et al.	2001
S74	Specification-based test oracles for reactive systems	Richardson et al.	1992
S75	Model-based testing of object-oriented systems	Rumpe	2003
S76	Aiding modular design and verification of safety-critical time-triggered systems by use of executable formal specifications	Sakurai et al.	2008
S77	An evaluation of a model-based testing method for information systems	Santos-Neto et al.	2008

Bibliography

- [1] Elsevier. Contributor roles taxonomy. <https://www.elsevier.com/authors/policies-and-guidelines/credit-author-statement>. Accessed: 2022-08-12.
- [2] Tuomas Jaanu, Maria Paasivaara, and Casper Lassenius. Effects of four distances on communication processes in global software projects. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 231–234, 2012. ISSN: 1949-3789.
- [3] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Information and Software Technology*, 70:204–219, 2016.
- [4] Elizabeth Bjarnason and Helen Sharp. The role of distances in requirements communication: a case study. *Requirements Engineering*, 22(1):1–26, 2017.
- [5] Benjamin Lesage, Stephen Law, and Iain Bate. TACO: An industrial case study of test automation for COverage. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pages 114–124. ACM, 2018.
- [6] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 6 Publisher: Springer US.
- [7] Zeinab Alizadeh Barmi, Amir Hossein Ebrahimi, and Robert Feldt. Alignment of requirements specification and testing: A systematic mapping study. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 476–485, 2011.

- [8] Pertti Karhapää, Alireza Haghightakhah, and Markku Oivo. What do we know about alignment of requirements engineering and software testing? In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, pages 354–363. Association for Computing Machinery, 2017.
- [9] Arilo C. Dias-Neto and Guilherme H. Travassos. A picture from the model-based testing area: Concepts, techniques, and challenges. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 80 of *Advances in Computers*, pages 45–120. Elsevier, 2010.
- [10] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012.
- [11] Fabiano Dalpiaz, Alessio Ferrari, Xavier Franch, and Cristina Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, 35(5):115–119, 2018. Conference Name: IEEE Software.
- [12] C.J. Neill and P.A. Laplante. Requirements engineering: the state of the practice. *IEEE Software*, 20(6):40–45, 2003. Conference Name: IEEE Software.
- [13] Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering*, 17(1):57–78, 2012.
- [14] Mohamad Kassab, Colin Neill, and Phillip Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014.
- [15] D. Méndez Fernández, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. de la Vara, and R. Wieringa. Naming the Pain in Requirements Engineering: Contemporary Problems, Causes, and Effects in Practice. *Empirical Software Engineering*, 22(5):2298–2338, October 2017. arXiv: 1611.10288.
- [16] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Washington, DC, USA, 3rd edition, 2014.
- [17] SEBoK. System verification — sebok,, 2021. [Online; accessed 18-August-2022].

-
- [18] Mika V. Mäntylä and Casper Lassenius. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):430–448, 2009. Conference Name: IEEE Transactions on Software Engineering.
- [19] Alp Toygar. A new asset type: Digital assets. *Journal of International Technology and Information Management*, 22(4):9, 2013.
- [20] Peter E. D. Love, Jingyang Zhou, Jane Matthews, and Harbin Luo. Systems information modelling: Enabling digital asset management. *Advances in Engineering Software*, 102:155–165, 2016.
- [21] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches: a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, WEASELTech '07, pages 31–36. Association for Computing Machinery, 2007.
- [22] Michael Felderer, Philipp Zech, Ruth Breu, Matthias Büchler, and Alexander Pretschner. Model-based security testing: a taxonomy and systematic classification. *Software Testing, Verification and Reliability*, 26(2):119–148, 2016.
- [23] Goparaju Purna Sudhakar. A model of critical success factors for software projects. *Journal of Enterprise Information Management*, 25(6):537–558, 2012. Publisher: Emerald Group Publishing Limited.
- [24] Ewelina Kania, Elżbieta Radziszewska-Zielina, and Grzegorz Śladowski. Communication and information flow in polish construction projects. *Sustainability*, 12(21):9182, 2020.
- [25] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. Requirements are slipping through the gaps — a case study on causes effects of communication gaps in large-scale software development. In *2011 IEEE 19th International Requirements Engineering Conference*, pages 37–46, 2011]. ISSN: 1090-705X.
- [26] Charlene Xie, Dash Wu, Jianwen Luo, and Xiaoling Hu. A case study of multi-team communications in construction design under supply chain partnering. *Supply Chain Management: An International Journal*, 15(5):363–370, 2010. Publisher: Emerald Group Publishing Limited.
- [27] Sean D. Vermillion and Richard J. Malak. An investigation on requirement and objective allocation strategies using a principal-agent model. *Systems Engineering*, 23(1):100–117, 2020. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21511>.

BIBLIOGRAPHY

- [28] Rick Makkinga, Robin de Graaf, and Hans Voordijk. Successful verification of subcontracted work in the construction industry. *Systems Engineering*, 21(2):131–140, 2018. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21425>.
- [29] Azad M. Madni and Michael Sievers. Systems integration: Key perspectives, experiences, and challenges. *Systems Engineering*, 17(1):37–51, 2014. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21249>.
- [30] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2008. Publisher: BCS Learning & Development.
- [31] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, ver. 2.3 ebse technical report. ebse, 2007.
- [32] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, pages 1–10. ACM Press, 2014.
- [33] Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(2):77–131, 2007. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.344>.
- [34] Klaas-Jan Stol and Brian Fitzgerald. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology*, 27(3):1–51, 2018.
- [35] Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Inc., 2012.
- [36] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.
- [37] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [38] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

- [39] Michael Unterkalmsteiner. Early requirements traceability with domain-specific taxonomies - a pilot experiment. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 322–327, 2020. ISSN: 2332-6441.
- [40] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, 2008.
- [41] Sofia Charalampidou, Apostolos Ampatzoglou, Evangelos Karountzos, and Paris Avgeriou. Empirical studies on software traceability: A mapping study. *Journal of Software: Evolution and Process*, 33(2):e2294, February 2021. Publisher: John Wiley & Sons, Ltd.
- [42] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative Inquiry*, 12(2):219–245, 2006. Publisher: SAGE Publications Inc.
- [43] Vahid Garousi and Junji Zhi. A survey of software testing practices in canada. *Journal of Systems and Software*, 86(5):1354–1376, 2013.
- [44] Waleed Abdeen, Xingru Chen, and Michael Unterkalmsteiner. An approach for performance requirements verification and test environments generation. *Requirements Engineering*, 2022.
- [45] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, Jonathan Maletic, and Patrick Mäder. Traceability fundamentals. In *Software and systems traceability*, pages 3–22. Springer, 2012.
- [46] Hermann Kaindl. The missing link in requirements engineering. *ACM SIGSOFT Software Engineering Notes*, 18(2):30–39, April 1993.
- [47] Theresia Ratih Dewi Saputri and Seok-Won Lee. Ensuring traceability in modeling requirement using ontology based approach. In Seok-Won Lee and Takako Nakatani, editors, *Requirements Engineering Toward Sustainable World*, Communications in Computer and Information Science, pages 3–17. Springer, 2016.
- [48] Marina Murtazina and Tatiana Avdeenko. An ontology-based approach to the agile requirements engineering. In Nikolaj Bjørner, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of System Informatics*, Lecture Notes in Computer Science, pages 205–213. Springer International Publishing, 2019.
- [49] Gouri Deshpande, Quim Motger, Cristina Palomares, Ikagarjot Kamra, Katarzyna Biesialska, Xavier Franch, Guenther Ruhe, and Jason Ho. Requirements Dependency Extraction by Integrating Active Learning with

- Ontology-Based Retrieval. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 78–89, August 2020. ISSN: 2332-6441.
- [50] Alberto Sardinha, Yijun Yu, Nan Niu, and Awais Rashid. EA-tracer: identifying traceability links between code aspects and early aspects. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12*, pages 1035–1042, New York, NY, USA, March 2012. Association for Computing Machinery.
- [51] Mateusz Wieloch, Sorawit Amornborvornwong, and Jane Cleland-Huang. Trace-by-classification: A machine learning approach to generate trace links for frequently occurring software artifacts. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 110–114, May 2013. ISSN: 2157-2194.
- [52] Richard Torkar, Tony Gorschek, Robert Feldt, Mikael Svahnberg, Uzair Akbar Raja, and Kashif Kamran. REQUIREMENTS TRACEABILITY: A SYSTEMATIC REVIEW AND INDUSTRY CASE STUDY. *International Journal of Software Engineering and Knowledge Engineering*, 22(3):385–433, 2012.
- [53] Elke Bouillon, Patrick Mäder, and Ilka Philippow. A survey on usage scenarios for requirements traceability in practice. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 158–173. Springer, 2013.
- [54] O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, 1994.
- [55] P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *13th International Conference on Requirements Engineering*, pages 385–389, August 2005.
- [56] Fabian Fagerholm, Michael Felderer, Davide Fucci, Michael Unterkalmsteiner, Bogdan Marculescu, Markus Martini, Lars Göran Wallgren Tengberg, Robert Feldt, Bettina Lehtelä, Balázs Nagyvárad, and Jehan Khattak. Cognition in software engineering: A taxonomy and survey of a half-century of research. *ACM Computing Surveys*, 2021. Just Accepted.
- [57] M. M. Lehman. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, 1:213–221, 1979.
- [58] Manny M Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124. Springer, 1996.

- [59] Ehsan Zabardast, Julian Frattini, Javier Gonzalez-Huerta, Daniel Mendez, Tony Gorschek, and Krzysztof Wnuk. Assets in software engineering: What are they after all? *Journal of Systems and Software*, page 111485, 2022.
- [60] D. Janzen and H. Saiedian. Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, 2005. Conference Name: Computer.
- [61] Ian Molyneaux. *The Art of Application Performance Testing: From Strategy to Tools*. "O'Reilly Media, Inc.", 2014-12-15. Google-Books-ID: 187UBQAAQBAJ.
- [62] C.U. Smith and L.G. Williams. Software performance engineering: a case study including performance comparison with design alternatives. *IEEE Transactions on Software Engineering*, 19(7):720–741, July 1993. Conference Name: IEEE Transactions on Software Engineering.
- [63] Wikipedia. Healthcare.gov. <https://en.wikipedia.org/wiki/HealthCare.gov>. Accessed: 2020-03-15.
- [64] Cigniti Technologies. Classic cases where performance testing failures plagued large organizations. <https://www.cigniti.com/blog/2-classic-cases-where-performance-testing-failures-plague-large-organisations/>. Accessed: 2020-03-15.
- [65] Lawrence Chung, Brian A. Nixon, Eric Yu, and John Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer Science & Business Media, 2012. Google-Books-ID: MNrcBwAAQBAJ.
- [66] P.C. Clements. Coming attractions in software architecture. In *Proceedings of 5th International Workshop on Parallel and Distributed Real-Time Systems and 3rd Workshop on Object-Oriented Real-Time Systems*, pages 2–9, 1997.
- [67] Connie U. Smith and Lloyd G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2001. Google-Books-ID: X5VIQgAACAAJ.
- [68] Edward Moyer. For pokemon go, it's stop – at least temporarily. <https://www.cnet.com/news/for-pokemon-go-its-stop-at-least-temporarily/>. Accessed: 2019-10-10.
- [69] Murray Woodside, Greg Franks, and Dorina C. Petriu. The future of software performance engineering. In *Future of Software Engineering (FOSE '07)*, pages 171–187, 2007.
- [70] Bill Hasling, Helmut Goetz, and Klaus Beetz. Model based testing of system requirements using UML use case models. In *and Validation 2008 1st International Conference on Software Testing, Verification*, pages 367–376, 2008. ISSN: 2159-4848.

BIBLIOGRAPHY

- [71] Florian Häser, Michael Felderer, and Ruth Breu. Software paradigms, assessment types and non-functional requirements in model-based integration testing: a systematic literature review. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, pages 1–10. ACM Press, 2014.
- [72] Mark Utting, Alexander Pretschner, Bruno Legeard, C. Mark Utting, Er Pretschner, Bruno Legeard, Mark Uttinga, Er Pretschnerb, and Bruno Legeardc. Legeard b., a taxonomy of model-based testing. *Department of Computer Science, The University of Waikato, Hamilton, New Zealand*, 2006.
- [73] Alessio Ferrari, Giorgio O Spagnolo, and Stefania Gnesi. Towards a dataset for natural language requirements processing. In *23rd International Workshop on Requirements Engineering Foundation for Software Quality Workshops (REFSQ)*, page 6, 2017.
- [74] Rafa E Al-Qutaihs. Quality models in software engineering literature: An analytical and comparative study. *Journal of American Science*, page 10, 2010.
- [75] Khashayar Khosravi and Yann-Gaël Guéhéneuc. A quality model for design patterns. *German Industry Standard*, 2004.
- [76] Francois Coallier. Software engineering–product quality–part 1: Quality model. *International Organization for Standardization: Geneva, Switzerland*, 2001.
- [77] Robert B. Grady and Deborah L. Caswell. *Software metrics: establishing a company-wide program*. Prentice-Hall, 1987.
- [78] ISO. Software product quality model - iso25010. <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>. Accessed: 2019-09-12.
- [79] B.W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1):75–88, 1984.
- [80] R.G. Dromey. A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2):146–162, February 1995. Conference Name: IEEE Transactions on Software Engineering.
- [81] Jim A McCall, Paul K Richards, and Gene F Walters. Factors in software quality, volumes i, ii, and iii. *US Rome Air Development Center Reports, US Department of Commerce, USA*, 1977.
- [82] Ina Schieferdecker. Model-based testing. *IEEE Software*, 29(1):14–18, 2012. Conference Name: IEEE Software.

-
- [83] Wenbin Li, Franck Le Gall, and Naum Spaseski. A survey on model-based testing tools for test case generation. In Vladimir Itsykson, Andre Scedrov, and Victor Zakharov, editors, *Tools and Methods of Program Analysis*, Communications in Computer and Information Science, pages 77–89. Springer International Publishing, 2018.
- [84] Mahmoud Abdelgawad, Sterling McLeod, Anneliese Andrews, and Jing Xiao. Model-based testing of a real-time adaptive motion planning system. *Advanced Robotics*, 31(22):1159–1176, 2017.
- [85] Alexander Pretschner, Wolfgang Prenninger, Stefan Wagner, Christian Kühnel, Martin Baumgartner, Bernd Sostawa, Rüdiger Zölch, and Thomas Stauner. One evaluation of model-based testing and its automation. In *Proceedings of the 27th International Conference on Software engineering*, pages 392–401, 2005.
- [86] Glenford J. Myers. *The Art of Software Testing, Second Edition*. John Wiley & Sons, 2004.
- [87] Amit Paradkar, K.C. Tai, and M.A. Vouk. Specification-based testing using cause-effect graphs. *Annals of Software Engineering*, 4(1):133–157, 1997.
- [88] Maicon Bernardino da Silveira, Elder de M Rodrigues, Avelino F Zorzo, Leandro T Costa, Hugo V Vieira, and Flávio Moreira de Oliveira. Generation of scripts for performance testing based on uml models. In *The 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 258–263, 2011.
- [89] Ravinder Veer Hooda. A future approach for model-based testing: Issues and guidelines. *International Journal of Latest Research in Science and Technology*, 2(1):541–543, 2013.
- [90] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. The value of mapping studies – a participant-observer case study. In *14th International Conference on Evaluation and Assessment in Software Engineering (EASE) (EASE)*, 2010. Publisher: BCS Learning & Development.
- [91] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An upyear. *Information and Software Technology*, 64:1–18, 2015.
- [92] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2016. Google-Books-ID: bQtQDQAAQBAJ.
- [93] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaïel. Model-based conformance testing of WS-BPEL compositions. In *2012 IEEE 36th Annual*

- Computer Software and Applications Conference Workshops*, pages 452–457, 2012.
- [94] Fredrik Abbors, Tanwir Ahmad, Dragos Truscan, and Ivan Porres. Model-based performance testing in the cloud using the mbpet tool. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, ICPE '13*, pages 423–424. Association for Computing Machinery, 2013.
- [95] Alessandro Faedo. Natural language requirements dataset. Institute of Information Science and Technologies. <http://fmt.isti.cnr.it/nlreqdataset/>. Accessed: 2019-02-08.
- [96] Waleed Abdeen, Xingru Chen, and Michael Unterkalmsteiner. Model-Based Testing for Performance Requirements Dataset, June 2021.
- [97] Andreas Johnsen, Kristina Lundqvist, Kaj Hänninen, Paul Pettersson, and Martin Torelm. Experience report: Evaluating fault detection effectiveness and resource efficiency of the architecture quality assurance framework and tool. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*, pages 271–281, 2017. ISSN: 2332-6549.
- [98] Afef Jmal Maâlej, Manel Hamza, Moez Krichen, and Mohamed Jmaïel. Automated significant load testing for WS-BPEL compositions. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 144–153, 2013.
- [99] Richard Schumi, Priska Lang, Bernhard K Aichernig, Willibald Krenn, and Rupert Schlick. Checking response-time properties of web-service applications under stochastic user profiles. In *IFIP International Conference on Testing Software and Systems*, pages 293–310. Springer, 2017.
- [100] Mustafa Al-tekreeti, Kshirasagar Naik, Atef Abdrabou, Marzia Zaman, and Pradeep Srivastava. Test generation for performance evaluation of mobile multimedia streaming applications:. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development*, pages 225–236. SCITEPRESS - Science and Technology Publications, 2018.
- [101] Vahid Garousi. Fault-driven stress testing of distributed real-time software based on UML models. *Software Testing, Verification and Reliability*, 21(2):101–124, 2011.
- [102] Alessio Gambi, Antonio Filieri, and Schahram Dustdar. Iterative test suites refinement for elastic computing systems. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013*, pages 635–638. Association for Computing Machinery, 2013.

-
- [103] Deepak Gangadharan, Samarjit Chakraborty, and Roger Zimmermann. Fast model-based test case classification for performance analysis of multimedia MPSoC platforms. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 413–422. Association for Computing Machinery, 2009.
- [104] Helge Löding and Jan Peleska. Timed moore automata: Test data generation and model checking. In *Verification and Validation 2010 Third International Conference on Software Testing*, pages 449–458, 2010. ISSN: 2159-4848.
- [105] Claas Wilke, Sebastian Götz, Jan Reimann, and Uwe Aßmann. Vision paper: Towards model-based energy testing. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, Lecture Notes in Computer Science, pages 480–489. Springer, 2011.
- [106] Danny Weyns. Towards an integrated approach for validating qualities of self-adaptive systems. In *Proceedings of the Ninth International Workshop on Dynamic Analysis*, WODA 2012, pages 24–29. Association for Computing Machinery, 2012.
- [107] Junyi Wang, Xiaoying Bai, Linyi Li, Zhicheng Ji, and Haoran Ma. A model-based framework for cloud API testing. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 60–65, 2017. ISSN: 0730-3157.
- [108] Padma Iyengar, Michael Spieker, Pablo Tecker, Juergen Wuebbelmann, Clemens Westerkamp, Walter van der Heiden, and Andreas Willert. Applicability of an integrated model-based testing approach for rtcs. In *2011 9th IEEE International Conference on Industrial Informatics*, pages 871–876. IEEE, 2011.
- [109] Lars Luthmann, Andreas Stephan, Johannes Bürdek, and Malte Lochau. Modeling and testing product lines with unbounded parametric real-time constraints. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A*, SPLC '17, pages 104–113. Association for Computing Machinery, 2017.
- [110] Wenbin Li, Franck Le Gall, Panagiotis Vlachreas, and Alexey Cheptsov. Quality assurance for component-based systems in embedded environments. In *2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC)*, pages 171–176, 2018.
- [111] Mehrdad Saadatmand and Mikael Sjödin. Testing of timing properties in real-time systems: Verifying clock constraints. In *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, volume 2, pages 152–158, 2013. ISSN: 1530-1362.

- [112] Maicon Bernardino, Avelino F. Zorzo, and Elder M. Rodrigues. Canopus: A domain-specific language for modeling performance testing. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 157–167, 2016.
- [113] Valentin Chimisliu and Franz Wotawa. Abstracting timing information in UML state charts via temporal ordering and LOTOS. In *Proceedings of the 6th International Workshop on Automation of Software Test, AST '11*, pages 8–14. Association for Computing Machinery, 2011.
- [114] Sebastian Siegl, Martin Russer, and Kai-Steffen Hielscher. Partitioning the requirements of embedded systems by input/output dependency analysis for compositional creation of parallel test models. In *2015 Annual IEEE Systems Conference (SysCon) Proceedings*, pages 96–102, 2015.
- [115] Elder Rodrigues, Maicon Bernardino, Leandro Costa, Avelino Zorzo, and Flavio Oliveira. PLeTsPerf - a model-based performance testing tool. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–8, 2015. ISSN: 2159-4848.
- [116] Eduard Paul Enoiu, Daniel Sundmark, and Paul Pettersson. Model-based test suite generation for function block diagrams using the UPPAAL model checker. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 158–167, 2013.
- [117] Afef Jmal Maâlej, Moez Krichen, and Mohamed Jmaïel. Conformance testing of WS-BPEL compositions under various load conditions. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 371–371, 2012. ISSN: 0730-3157.
- [118] Juri Vain, Leonidas Tsiopoulos, Vyacheslav Kharchenko, Apneet Kaur, Maksim Jenihhin, and Jaan Raik. Multi-fragment markov model guided online test generation for MPSoC. In *ICTERI 2017 proceedings*, page 14, 2017.
- [119] Chunhui Wang, Fabrizio Pastore, and Lionel Briand. System testing of timing requirements based on use cases and timed automata. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 299–309, 2017.
- [120] Matteo Camilli, Angelo Gargantini, Patrizia Scandurra, and Carlo Belletini. Event-based runtime verification of temporal properties using time basic petri nets. In Clark Barrett, Misty Davies, and Temesghen Kahsai, editors, *NASA Formal Methods*, Lecture Notes in Computer Science, pages 115–130. Springer International Publishing, 2017.

- [121] Wolfgang Prenninger, Mohammad El-Ramly, and Marc Horstmann. 15 case studies. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems*, volume 3472, pages 439–461. Springer Berlin Heidelberg, 2005. Series Title: Lecture Notes in Computer Science.
- [122] Muhammad Zohaib Iqbal, Andrea Arcuri, and Lionel Briand. Environment modeling and simulation for automated testing of soft real-time embedded software. *Software & Systems Modeling*, 14(1):483–524, February 2015.
- [123] Fredrik Abbors and Dragoş Truşcan. Approaching performance testing from a model-based testing perspective. In *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*, pages 125–128, 2010.
- [124] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer Science & Business Media, 2012. Google-Books-ID: QPVsM1-U8nkC.
- [125] William R Elmendorf. *Cause-effect graphs in functional testing*. IBM Poughkeepsie Laboratory, 1973.
- [126] Chih-Wei Ho, M.J. Johnson, L. Williams, and E.M. Maximilien. On agile performance requirements specification and testing. In *AGILE 2006 (AGILE'06)*, pages 6 pp.–52, 2006.
- [127] André B. Bondi. Best practices for writing and managing performance requirements: a tutorial. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, ICPE '12*, pages 1–8. Association for Computing Machinery, 2012.
- [128] Jonas Eckhardt, Andreas Vogelsang, Henning Femmer, and Philipp Mager. Challenging incompleteness of performance requirements by sentence patterns. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 46–55, 2016. ISSN: 2332-6441.
- [129] B.A. Nixon. Management of performance requirements for information systems. *IEEE Transactions on Software Engineering*, 26(12):1122–1146, 2000. Conference Name: IEEE Transactions on Software Engineering.
- [130] Zhiming Cai and Eric Yu. Addressing performance requirements using a goal and scenario-oriented approach. In Anne Banks Pidduck, M. Tamer Ozsu, John Mylopoulos, and Carson C. Woo, editors, *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 706–710. Springer, 2002.

BIBLIOGRAPHY

- [131] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004. Conference Name: IEEE Transactions on Software Engineering.
- [132] Mike Robinson and Liam Bannon. Questioning representations. In Liam Bannon, Mike Robinson, and Kjeld Schmidt, editors, *Proceedings of the Second European Conference on Computer-Supported Cooperative Work EC-SCW '91*, pages 219–233. Springer Netherlands, 1991.
- [133] Amer Al-Rawas and Steve Easterbrook. COMMUNICATION PROBLEMS IN REQUIREMENTS ENGINEERING: A FIELD STUDY. In *First Westminster Conference on Professional Awareness in Software Engineering, Royal Society*, page 12, 1996.
- [134] Alistair A R Cockburn. Characterizing people as nonlinear, firstorder components in software development. In *The 4th International MultiConference on Systems, Cybernetics and Informatics*, page 12, 1999.
- [135] G. Melnik and F. Maurer. Direct verbal communication as a catalyst of agile knowledge sharing. In *Agile Development Conference*, pages 21–31, 2004.
- [136] INCOSE and Wiley. *INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities*. John Wiley & Sons, Incorporated, New York, 2015.
- [137] A. Terry Bahill and Steven J. Henderson. Requirements development, verification, and validation exhibited in famous failures. *Systems Engineering*, 8(1):1–14, 2005. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.20017>.
- [138] Office of Electricity. August 2003 blackout. <https://www.energy.gov/oe/services/electricity-policy-coordination-and-implementation/august-2003-blackout>, 2003. Accessed: 2021-03-11.
- [139] Summaira Malik, Muhammad Taqi, José Moleiro Martins, Mário Nuno Mata, João Manuel Pereira, and António Abreu. Exploring the relationship between communication and success of construction projects: The mediating role of conflict. *Sustainability*, 13(8):4513, 2021.
- [140] Grisca Liebel, Matthias Tichy, Eric Knauss, Oscar Ljungkrantz, and Gerald Stieglbauer. Organisation and communication problems in automotive requirements engineering. *Requirements Engineering*, 23(1):145–167, March 2018.

-
- [141] E Bjarnason, K Wnuk, and B Regnell. Requirements are slipping through the gaps—a case study on causes & effects of communication gaps in large-scale software development. In *2011 IEEE 19th international requirements engineering conference*, pages 37–46. IEEE, 2011.
- [142] D. E. Damian and D. Zowghi. The impact of stakeholders’ geographical distribution on managing requirements in a multi-site organization. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 319–328, 2002. ISSN: 1090-705X.
- [143] Daniela E. Damian and Didar Zowghi. RE challenges in multi-site software development organisations. *Requirements Engineering*, 8(3):149–160, 2003.
- [144] D. Damian. Stakeholders in global requirements engineering: Lessons learned from practice. *IEEE Software*, 24(2):21–27, 2007. Conference Name: IEEE Software.
- [145] E Bjarnason, P Runeson, M Borg, M Unterkalmsteiner, E Engström, B Regnell, G Sabaliauskaite, A Loconsole, T Gorschek, and R Feldt. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical software engineering*, 19(6):1809–1855, 2014.
- [146] S Hotomski, EB Charrada, and M Glinz. An exploratory study on handling requirements and acceptance test documentation in industry. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 116–125. IEEE, 2016.
- [147] A Terry Bahill and Steven J Henderson. Requirements development, verification, and validation exhibited in famous failures. *Systems engineering*, 8(1):1–14, 2005.
- [148] Massila Kamalrudin and Safiah Sidek. A review on software requirements validation and consistency management. *International journal of software engineering and its applications*, 9(10):39–58, 2015.
- [149] Hafiz Anas Bilal, Muhammad Ilyas, Qandeel Tariq, and Muhammad Humayun. Requirements validation techniques: An empirical study. *International Journal of Computer Applications*, 148(14), 2016.
- [150] Uzair Akbar Raja. Empirical studies of requirements validation techniques. In *2009 2nd International Conference on Computer, Control and Communication*, pages 1–9, 2009.
- [151] Alan B. Marchant. Obstacles to the flow of requirements verification. *Systems Engineering*, 13(1):1–13, 2010. [_eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.20127](https://onlinelibrary.wiley.com/doi/pdf/10.1002/sys.20127).

BIBLIOGRAPHY

- [152] P Runeson and M Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.
- [153] SEBoK. Stakeholder needs and requirements — sebok,, 2020. [Online; accessed 15-February-2021].
- [154] Paul Grünbacher and Norbert Seyff. Requirements Negotiation. In Aybüke Aurum and Claes Wohlin, editors, *Engineering and Managing Software Requirements*, pages 143–162. Springer, Berlin, Heidelberg, 2005.
- [155] Stephen C-Y. Lu and Nan Jing. A socio-technical negotiation approach for collaborative design in software engineering. *International Journal of Collaborative Engineering*, 1(1-2):185–209, 2009.
- [156] Daniela Damian, Remko Helms, Irwin Kwan, Sabrina Marczak, and Benjamin Koelewijn. The role of domain knowledge and cross-functional communication in socio-technical coordination. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 442–451. IEEE, 2013.
- [157] V. Pekar, M. Felderer, and R. Breu. Improvement methods for software requirement specifications: A mapping study. In *2014 9th International Conference on the Quality of Information and Communications Technology*, pages 242–245, 2014.
- [158] IEEE. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998. Conference Name: IEEE Std 830-1998.
- [159] M. Weber and J. Weisbrod. Requirements engineering in automotive development-experiences and challenges. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 331–340, 2002. ISSN: 1090-705X.
- [160] Peerasit Patanakul. Managing large-scale IS/IT projects in the public sector: Problems and causes leading to poor performance. *The Journal of High Technology Management Research*, 25(1):21–35, 2014.
- [161] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017.
- [162] Sourour Maalem and Nacereddine Zarour. Challenge of validation in requirements engineering. *Journal of Innovation in Digital Ecosystems*, 3(1):15–21, 2016.

-
- [163] J. Scheffczyk, U. M. Borghoff, A. Birk, and J. Siedersleben. Pragmatic consistency management in industrial requirements specifications. In *Third IEEE International Conference on Software Engineering and Formal Methods (SEFM'05)*, pages 272–281, 2005. ISSN: 2160-7656.
- [164] M. Kamalrudin. Automated software tool support for checking the inconsistency of requirements. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 693–697, 2009. ISSN: 1938-4300.
- [165] Thiago C de Sousa, Jorge R Almeida Jr, Sidney Viana, and Judith Pavón. Automatic analysis of requirements consistency with the b method. *ACM SIGSOFT Software Engineering Notes*, 35(2):4, 2010.
- [166] Nadzeya Kiyavitskaya, Nicola Zeni, Luisa Mich, and Daniel M. Berry. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. *Requirements Engineering*, 13(3):207–239, 2008.
- [167] Hui Yang, Alistair Willis, Anne De Roeck, and Bashar Nuseibeh. Automatic detection of nocuous coordination ambiguities in natural language requirements. In *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*, page 53. ACM Press, 2010.
- [168] Carlos Huertas, Manuel Gómez-Ruelas, Reyes Juárez-Ramírez, and Héctor Plata. A formal approach for measuring the lexical ambiguity degree in natural language requirement specification: Polysemes and homonyms focused. In *2011 International Conference on Uncertainty Reasoning and Knowledge Engineering*, pages 115–118, Aug 2011.
- [169] A. Ferrari and S. Gnesi. Using collective intelligence to detect pragmatic ambiguities. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 191–200, 2012. ISSN: 2332-6441.
- [170] T. Y. Chen, Pak-Lok Poon, Sau-Fun Tang, T. H. Tse, and Y. T. Yu. Towards a problem-driven approach to perspective-based reading. In *7th IEEE International Symposium on High Assurance Systems Engineering, 2002. Proceedings.*, pages 221–229, 2002. ISSN: 1530-2059.
- [171] T. Berling and P. Runeson. Evaluation of a perspective based review method applied in an industrial setting. *IEE Proceedings - Software*, 150(3):177–184, 2003. Publisher: IET Digital Library.
- [172] J. Polpinij. An ontology-based text processing approach for simplifying ambiguity of requirement specifications. In *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pages 219–226, 2009.

BIBLIOGRAPHY

- [173] H. Hu, L. Zhang, and C. Ye. Semantic-based requirements analysis and verification. In *2010 International Conference on Electronics and Information Engineering*, pages V1–241–V1–246, 2010.
- [174] A. Gross and J. Doerr. What you need is what you get!: The vision of view-based requirements specifications. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 171–180, 2012.
- [175] A. Bucchiarone, S. Gnesi, and P. Pierini. Quality analysis of NL requirements: an industrial case study. In *13th IEEE International Conference on Requirements Engineering (RE'05)*, pages 390–394, 2005. ISSN: 2332-6441.
- [176] I. Menzel, M. Mueller, A. Gross, and J. Doerr. An experimental comparison regarding the completeness of functional requirements specifications. In *2010 18th IEEE International Requirements Engineering Conference*, pages 15–24, 2010. ISSN: 2332-6441.
- [177] JungWon Byun, SungYul Rhew, ManSoo Hwang, Vijayan Sugumara, SooYong Park, and SooJin Park. Metrics for measuring the consistencies of requirements with objectives and constraints. *Requirements Engineering*, 19(1):89–104, 2014.
- [178] Julio Cesar Sampaio do Prado Leite and Peter Freeman. Requirements validation through viewpoint resolution. *IEEE Trans. Software Eng.*, 17(12):1253–1269, 1991.
- [179] M. Bano. Addressing the challenges of requirements ambiguity: A review of empirical literature. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 21–24, 2015. ISSN: 2329-6356.
- [180] Gauthier Fanmuy, Anabel Fraga, and Juan Llorens. Requirements verification in the industry. In Omar Hammami, Daniel Krob, and Jean-Luc Voiron, editors, *Complex Systems Design & Management*, pages 145–160. Springer, 2012.
- [181] Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through—a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/spip.253>.
- [182] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. An exploratory study on handling requirements and acceptance test documentation in industry. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 116–125, 2016. ISSN: 2332-6441.

-
- [183] Namfon Assawamekin, Thanwadee Sunetnanta, and Charnyote Pluempitiwiriyawej. Ontology-based multiperspective requirements traceability framework. *Knowledge and Information Systems*, 25(3):493–522, 2010.
- [184] Yonghua Li and Jane Cleland-Huang. Ontology-based trace retrieval. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 30–36, 2013. ISSN: 2157-2194.
- [185] M. Hoffmann, N. Kuhn, M. Weber, and M. Bittner. Requirements for requirements management tools. In *Proceedings. 12th IEEE International Requirements Engineering Conference, 2004.*, pages 301–308, 2004. ISSN: 1090-705X.
- [186] IBM. Engineering requirements management doors documentation. <https://www.ibm.com/docs/en/ermd/9.6.0?topic=links-creating-external>, 2021. Accessed: 2022-03-09.
- [187] Tovi Grossman, George Fitzmaurice, and Ramtin Attar. A survey of software learnability: metrics, methodologies and guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09*, pages 649–658. Association for Computing Machinery, 2009.
- [188] D.E. Damian and D. Zowghi. An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, page 10 pp. IEEE, 2003.
- [189] Artem Katasonov and Markku Sakkinen. Requirements quality control: a unifying framework. *Requirements Engineering*, 11(1):42–57, 2006.
- [190] V. Ambriola and V. Gervasi. Processing natural language requirements. In *Proceedings 12th IEEE International Conference Automated Software Engineering*, pages 36–45, 1997.
- [191] Francis Chantree, Bashar Nuseibeh, Anne de Roeck, and Alistair Willis. Identifying nocuous ambiguities in natural language requirements. In *14th IEEE International Requirements Engineering Conference (RE'06)*, pages 59–68, 2006. ISSN: 2332-6441.
- [192] Jean C Bedard, Cynthia Jackson, Michael L Ettredge, and Karla M Johnstone. The effect of training on auditors' acceptance of an electronic work system. *International Journal of Accounting Information Systems*, 4(4):227–250, 2003.
- [193] Nurmazilah Mahzan and Andy Lymer. Examining the adoption of computer-assisted audit tools and techniques: Cases of generalized audit software use

BIBLIOGRAPHY

- by internal auditors. *Managerial Auditing Journal*, 29(4):327–349, 2014. Publisher: Emerald Group Publishing Limited.
- [194] Hannah Gascho Rempel and Margaret Mellinger. Bibliographic management tool adoption and use a qualitative research study using the UTAUT model. *Reference & User Services Quarterly*, 54(4):43–53, 2015. Number: 4.
- [195] Samuel Laryea. Subcontract and supply enquiries in the tender process of contractors. *Construction Management and Economics*, page 13, 2009.
- [196] Suzanne Robertson and James Robertson. *Mastering the requirements process: Getting requirements right*. Addison-wesley, 2012.
- [197] Albert P.C. Chan and Goodenough D. Opong. Managing the expectations of external stakeholders in construction projects. *Engineering, Construction and Architectural Management*, 24(5):736–756, 2017. Publisher: Emerald Publishing Limited.
- [198] Ka Yan Mok, Geoffrey Qiping Shen, and Jing Yang. Stakeholder management studies in mega construction projects: A review and future directions. *International Journal of Project Management*, 33(2):446–457, 2015.
- [199] Stefan Olander. Stakeholder impact analysis in construction project management. *Construction Management and Economics*, 25(3):277–287, 2007. Publisher: Routledge eprint: <https://doi.org/10.1080/01446190600879125>.
- [200] Arun A. Elias, Robert Y. Cavana, and Laurie S. Jackson. Stakeholder analysis for r&d project management. *R&D Management*, 32(4):301–310, 2002. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9310.00262>.
- [201] SEBoK. Uk west coast route modernisation project — sebok,, 2020. [Online; accessed 5-April-2021].
- [202] Vahid Garousi, Kai Petersen, and Baris Ozkan. Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Information and Software Technology*, 79:106–127, 2016.
- [203] T Gorschek, P Garre, S Larsson, and C Wohlin. A model for technology transfer in practice. *IEEE software*, 23(6):88–95, 2006.
- [204] D. Rombach and R. Achatz. Research collaborations between academia and industry. In *FoSE 2007: Future of Software Engineering*, pages 29–36, 2007.
- [205] A.M. Connor, J. Buchan, and K. Petrova. Bridging the research-practice gap in requirements engineering through effective teaching and peer learning. In *ITNG 2009 - 6th International Conference on Information Technology: New Generations*, pages 678–683, 2009.

-
- [206] A. Sandberg, L. Pareto, and T. Arts. Agile collaborative research: Action principles for industry-academia collaboration. *IEEE Software*, 28(4):74–83, 2011.
- [207] P. Runeson and S. Minör. The 4+1 view model of industry-academia collaboration. In *WISE 2014 - Proceedings of the 2014 ACM International Workshop on Long-Term Industrial Collaboration on Software Engineering, Co-located with ASE 2014*, pages 21–24, 2014.
- [208] Patrick Mäder and Alexander Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, April 2015.
- [209] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006. Conference Name: IBM Systems Journal.
- [210] Balasubramaniam Ramesh, Curtis Stubbs, Timothy Powers, and Michael Edwards. Requirements traceability: Theory and practice. *Annals of Software Engineering*, 3(1):397–415, 1997.
- [211] Hannes Schwarz, Jürgen Ebert, and Andreas Winter. Graph-based traceability: a comprehensive approach. *Software & Systems Modeling*, 9(4):473–492, 2010.
- [212] Patrick Mader, Orlena Gotel, and Ilka Philippow. Motivation Matters in the Traceability Trenches. In *2009 17th IEEE International Requirements Engineering Conference*, pages 143–148, August 2009. ISSN: 2332-6441.
- [213] Orlena Gotel and Patrick Mäder. Acquiring tool support for traceability. In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 43–68. Springer London, 2012.
- [214] Salome Maro, Jan-Philipp Steghöfer, and Miroslaw Staron. Software traceability in the automotive domain: Challenges and solutions. *Journal of Systems and Software*, 141:85–110, 2018.
- [215] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.
- [216] Michael Unterkalmsteiner. Early requirements traceability with domain-specific taxonomies—a pilot experiment. In *28th International Requirements Engineering Conference (RE)*, pages 322–327. IEEE, 2020.

BIBLIOGRAPHY

- [217] Michael Unterkalmsteiner. TT-RecS: The Taxonomic Trace Recommender System. In *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 18–21, September 2020.
- [218] Javier Portillo-Rodríguez, Aurora Vizcaíno, Mario Piattini, and Sarah Beecham. Tools used in global software engineering: A systematic mapping review. *Information and Software Technology*, 54(7):663–685, 2012.
- [219] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. Data scientists in software teams: State of the art and challenges. *IEEE Transactions on Software Engineering*, 44(11):1024–1038, 2017.
- [220] Frâncila Weidt Neiva, José Maria N. David, Regina Braga, and Fernanda Campos. Towards pragmatic interoperability to support collaboration: A systematic review and mapping of the literature. *Information and Software Technology*, 72:137–150, 2016.
- [221] Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. A theory of distances in software engineering. *Information and Software Technology*, 70:204–219, 2016.
- [222] Dang Viet Dzung and Atsushi Ohnishi. Improvement of Quality of Software Requirements with Requirements Ontology. In *9th International Conference on Quality Software*, pages 284–289, August 2009.
- [223] Leonid Kof, Ricardo Gacitua, Mark Rouncefield, and Peter Sawyer. Ontology and Model Alignment as a Means for Requirements Validation. In *4th International Conference on Semantic Computing*, pages 46–51, September 2010.
- [224] Thomas Moser, Dietmar Winkler, Matthias Heindl, and Stefan Biffl. Requirements Management with Semantic Technology: An Empirical Study on Automated Requirements Categorization and Conflict Analysis. In *23rd International Conference on Advanced Information Systems Engineering*, pages 3–17, London, UK, 2011. Springer.
- [225] Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09*, pages 1–12, New York, NY, USA, May 2009. Association for Computing Machinery.
- [226] Roel Wieringa. Design science methodology: principles and practice. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, volume 2, page 493, Cape Town, South Africa, 2010. ACM Press.

-
- [227] Jane Webster and Richard T. Watson. Analyzing the past to prepare for the future: Writing a literature review. *MIS Quarterly*, 26(2):xiii–xxiii, 2002. Publisher: Management Information Systems Research Center, University of Minnesota.
- [228] Juergen Rilling, René Witte, Yonggang Zhang, and Yonggang Zhang. Automatic traceability recovery: An ontological approach. page 13, 2007.
- [229] Diego Dermeval, Jéssyka Vilela, Ig Ibert Bittencourt, Jaelson Castro, Seiji Isotani, Patrick Brito, and Alan Silva. Applications of ontologies in requirements engineering: a systematic review of the literature. *Requirements Engineering*, 21(4):405–437, 2016.
- [230] Lars Marius Garshol. Metadata? thesauri? taxonomies? topic maps! making sense of it all. *Journal of Information Science*, 30(4):378–391, 2004.
- [231] Jun Lin, Chan Chou Lin, Jane Cleland-Huang, Raffaella Settini, Joseph Amaya, Grace Bedford, Brian Berenbach, Oussama Ben Khadra, Chuan Duan, and Xuchang Zou. Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability. In *14th IEEE International Requirements Engineering Conference (RE’06)*, pages 363–364, September 2006. ISSN: 2332-6441.
- [232] A. Marcus and J.I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 125–135, May 2003. ISSN: 0270-5257.
- [233] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 306–315, September 2004. ISSN: 1063-6773.
- [234] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. ADAMS Re-Trace: a traceability recovery tool. In *Ninth European Conference on Software Maintenance and Reengineering*, pages 32–41, March 2005. ISSN: 1534-5351.
- [235] J.H. Hayes, A. Dekhtyar, and S.K. Sundaram. Improving after-the-fact tracing and mapping: supporting software quality predictions. *IEEE Software*, 22(6):30–37, November 2005. Conference Name: IEEE Software.
- [236] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Can Information Retrieval Techniques Effectively Support Traceability Link Recovery? In *14th IEEE International Conference on Program Comprehension (ICPC’06)*, pages 307–316, June 2006. ISSN: 1092-8138.

BIBLIOGRAPHY

- [237] M. Lormans and A. van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 10 pp.–56, March 2006. ISSN: 1534-5351.
- [238] A. De Lucia, R. Oliveto, F. Zurolo, and M. Di Penta. Improving Comprehensibility of Source Code via Traceability Information: a Controlled Experiment. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 317–326, June 2006. ISSN: 1092-8138.
- [239] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. Feature location via information retrieval based filtering of a single scenario execution trace. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE '07*, pages 234–243, New York, NY, USA, November 2007. Association for Computing Machinery.
- [240] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, 16(4):13–es, September 2007.
- [241] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering*, 14(1):57–92, February 2009.
- [242] Collin McMillan, Denys Poshyvanyk, and Meghan Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 41–48, May 2009. ISSN: 2157-2194.
- [243] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. On the role of the nouns in IR-based traceability recovery. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 148–157, May 2009. ISSN: 1092-8138.
- [244] Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. The role of the coverage analysis during IR-based traceability recovery: A controlled experiment. In *2009 IEEE International Conference on Software Maintenance*, pages 371–380, September 2009. ISSN: 1063-6773.
- [245] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Traceability Recovery Using Numerical Analysis. In *2009 16th Working Conference on Reverse Engineering*, pages 195–204, October 2009. ISSN: 2375-5369.
- [246] Nouh Alhindawi, Omar Meqdadi, Brian Bartman, and Jonathan I. Maletic. A tracelab-based solution for identifying traceability links using LSI. In *2013*

-
- 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 79–82, San Francisco, CA, USA, May 2013. IEEE.
- [247] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation. *Information and Software Technology*, 55(4):741–754, April 2013.
- [248] Tathagata Dasgupta, Mark Grechanik, Evan Moritz, Bogdan Dit, and Denys Poshyvanyk. Enhancing Software Traceability by Automatically Expanding Corpora with Relevant Documentation. In *2013 IEEE International Conference on Software Maintenance*, pages 320–329, September 2013. ISSN: 1063-6773.
- [249] Sugandha Lohar, Sorawit Amornborvornwong, Andrea Zisman, and Jane Cleland-Huang. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013*, pages 378–388, New York, NY, USA, August 2013. Association for Computing Machinery.
- [250] Gabriele Bavota, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, Fabio Ricci, and Genoveffa Tortora. The role of artefact corpus in LSI-based traceability recovery. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 83–89, May 2013. ISSN: 2157-2194.
- [251] Patrick Rempel, Patrick Mäder, and Tobias Kuschke. Towards feature-aware retrieval of refinement traces. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 100–104, May 2013. ISSN: 2157-2194.
- [252] Diana Diaz, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Silvia Takahashi, and Andrea De Lucia. Using code ownership to improve IR-based Traceability Link Recovery. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 123–132, May 2013. ISSN: 1092-8138.
- [253] Gabriele Bavota, Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. Enhancing software artefact traceability recovery processes with link count information. *Information and Software Technology*, 56(2):163–182, February 2014.
- [254] Anas Mahmoud and Nan Niu. On the role of semantics in automated requirements tracing. *Requirements Engineering*, 20(3):281–300, September 2015.

BIBLIOGRAPHY

- [255] Giuliano Antoniol, Gerardo Canfora, A. Lucia, and G. Casazza. Information Retrieval Models for Recovering Traceability Links between Code and Documentation. *Software Maintenance, IEEE International Conference on*, 0:40, January 2000.
- [256] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, October 2002. Conference Name: IEEE Transactions on Software Engineering.
- [257] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proceedings. 7th International Workshop on Principles of Software Evolution, 2004.*, pages 49–54, September 2004. ISSN: 1550-4077.
- [258] Chuan Duan and Jane Cleland-Huang. Clustering support for automated tracing. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, ASE '07*, pages 244–253, New York, NY, USA, November 2007. Association for Computing Machinery.
- [259] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, E. Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, September 2007.
- [260] Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 133–142, September 2011. ISSN: 1063-6773.
- [261] Anas Mahmoud and Nan Niu. Source code indexing for automated tracing. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '11*, pages 3–9, New York, NY, USA, May 2011. Association for Computing Machinery.
- [262] Sandeep Pandanaboyana, Shreeram Sridharan, Jesse Yannelli, and Jane Huffman Hayes. REquirements TRacing on target (RETRO) enhanced with an automated thesaurus builder: An empirical study. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 61–67, 2013. ISSN: 2157-2194.
- [263] Annibale Panichella, Collin McMillan, Evan Moritz, Davide Palmieri, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. When and How Using Structural Information to Improve IR-Based Traceability Recovery. In

-
- 2013 17th European Conference on Software Maintenance and Reengineering*, pages 199–208, March 2013. ISSN: 1534-5351.
- [264] Anas Mahmoud and Nan Niu. Supporting requirements to code traceability through refactoring. *Requirements Engineering*, 19(3):309–329, September 2014.
- [265] Jin Guo, Natawut Monaikul, Cody Plepel, and Jane Cleland-Huang. Towards an intelligent domain-specific traceability solution. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 755–766, Vasteras Sweden, September 2014. ACM.
- [266] Annibale Panichella, Andrea De Lucia, and Andy Zaidman. Adaptive User Feedback for IR-Based Traceability Recovery. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 15–21, May 2015. ISSN: 2157-2194.
- [267] Yonghee Shin, Jane Huffman Hayes, and Jane Cleland-Huang. Guidelines for Benchmarking Automated Software Traceability Techniques. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 61–67, May 2015. ISSN: 2157-2194.
- [268] Mona Rahimi, William Goss, and Jane Cleland-Huang. Evolving Requirements-to-Code Trace Links across Versions of a Software System. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 99–109, October 2016.
- [269] Jin Guo, Marek Gibiec, and Jane Cleland-Huang. Tackling the term-mismatch problem in automated trace retrieval. *Empirical Software Engineering*, 22(3):1103–1142, June 2017.
- [270] Bangchao Wang, Rong Peng, Zhuo Wang, Xiaomin Wang, and Yuanbang Li. An Automated Hybrid Approach for Generating Requirements Trace Links. *International Journal of Software Engineering and Knowledge Engineering*, 30(07):1005–1048, July 2020.
- [271] Yonggang Zhang, René Witte, Juergen Rilling, and Volker Haarslev. An Ontology-based Approach for Traceability Recovery. *Concordia University*, page 15, 2006.
- [272] Namfon Assawamekin, Thanwadee Sunetnanta, and Charnyote Pluempitiwiriwaj. Resolving Multiperspective Requirements Traceability through Ontology Integration. In *2008 IEEE International Conference on Semantic Computing*, pages 362–369, August 2008.

BIBLIOGRAPHY

- [273] Namfon Assawamekin, Thanwadee Sunetnanta, and Charnyote Pluempitiwiriwaj. MUPRET: An Ontology-Driven Traceability Tool for Multiperspective Requirements Artifacts. In *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, pages 943–948, June 2009.
- [274] Jan Novacek, Ali Ahari, Alessandro Cornaglia, Frederik Haxel, Alexander Viehl, Oliver Bringmann, and Wolfgang Rosenstiel. Ontology-Supported Design Parameter Management for Change Impact Analysis. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 9–16, August 2018.
- [275] J.C. Caralt and J.W. Kim. Ontology driven requirements query. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007. ISSN: 1530-1605.
- [276] Fangshu Di and Maolin Zhang. An Improving Approach for Recovering Requirements-to-Design Traceability Links. In *2009 International Conference on Computational Intelligence and Software Engineering*, pages 1–6, December 2009.
- [277] Shinpei Hayashi, Takashi Yoshikawa, and Motoshi Saeki. Sentence-to-Code Traceability Recovery with Domain Ontologies. In *2010 Asia Pacific Software Engineering Conference*, pages 385–394, November 2010. ISSN: 1530-1362.
- [278] Muhammad Atif Javed, Srdjan Stevanetic, and Uwe Zdun. Towards a pattern language for construction and maintenance of software architecture traceability links. In *Proceedings of the 21st European Conference on Pattern Languages of Programs, EuroPlop '16*, pages 1–20, New York, NY, USA, July 2016. Association for Computing Machinery.
- [279] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana, and Paul Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, July 2004.
- [280] Waraporn Jirapanthong and Andrea Zisman. XTraQue: traceability for product line systems. *Software & Systems Modeling*, 8(1):117–144, February 2009.
- [281] Muhammad Shahid and Suhaimi Ibrahim. Change impact analysis with a software traceability approach to support software maintenance. In *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 391–396, January 2016. ISSN: 2151-1411.
- [282] A. von Knethen. Change-oriented requirements traceability. Support for evolution of embedded systems. In *International Conference on Software Maintenance, 2002. Proceedings.*, pages 482–485, Montreal, QC, Canada, 2002. IEEE.

- [283] Xuchang Zou, Raffaella Settini, and Jane Cleland-Huang. Phrasing in Dynamic Requirements Trace Retrieval. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, volume 1, pages 265–272, September 2006. ISSN: 0730-3157.
- [284] Leonid Kof, Ricardo Gacitua, Mark Rouncefield, and Pete Sawyer. Concept mapping as a means of requirements tracing. In *2010 Third International Workshop on Managing Requirements Knowledge*, pages 22–31, September 2010.
- [285] Yanyan Lin and Xiaofeng Zhou. A traceability approach to constructing feature model from use case models. In *2012 International Conference on Computer Science and Service System*, pages 545–548, 2012.
- [286] Lei Chen, Dandan Wang, Junjie Wang, and Qing Wang. Enhancing Un-supervised Requirements Traceability with Sequential Semantics. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 23–30, December 2019. ISSN: 2640-0715.
- [287] Shiva Nejati, Mehrdad Sabetzadeh, Davide Falessi, Lionel Briand, and Thierry Coq. A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. *Information and Software Technology*, 54(6):569–590, June 2012.
- [288] Jin Guo, Jane Cleland-Huang, and Brian Berenbach. Foundations for an expert system in domain-specific traceability. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 42–51, Rio de Janeiro-RJ, Brazil, July 2013. IEEE.
- [289] Jessica Díaz, Jennifer Pérez, and Juan Garbajosa. A model for tracing variability from features to product-line architectures: a case study in smart grids. *Requirements Engineering*, 20(3):323–343, September 2015.
- [290] Robert Andrei Buchmann and Dimitris Karagiannis. Modelling mobile app requirements for semantic traceability. *Requirements Engineering*, 22(1):41–75, March 2017.
- [291] Xuchang Zou, Raffaella Settini, and Jane Cleland-Huang. Improving automated requirements trace retrieval: a study of term-based enhancement methods. *Empirical Software Engineering*, 15(2):119–146, April 2010.
- [292] Rodrigo Perozzo Noll and Marcelo Blois Ribeiro. Ontological Traceability over the Unified Process. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 249–255, March 2007.

BIBLIOGRAPHY

- [293] Jane Cleland-Huang and Jin Guo. Towards more intelligent trace retrieval algorithms. In *Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering - RAISE 2014*, pages 1–6, Hyderabad, India, 2014. ACM Press.
- [294] Marc Eaddy, Alfred V. Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis. In *2008 16th IEEE International Conference on Program Comprehension*, pages 53–62, June 2008. ISSN: 1092-8138.
- [295] Raquel F. Lafet´ and Marcelo Maia. An Empirical Assessment of the Use of Execution Traces in Software Maintenance. In *2011 25th Brazilian Symposium on Software Engineering*, pages 154–163, September 2011.
- [296] Marcelo de Almeida Maia and Raquel Fialho Lafet´a. On the impact of trace-based feature location in the performance of software maintainers. *Journal of Systems and Software*, 86(4):1023–1037, April 2013.
- [297] Alexander Egyed and Paul Grnbacher. Supporting software understanding with automated requirements traceability. *International Journal of Software Engineering and Knowledge Engineering*, 15(05):783–810, October 2005. Publisher: World Scientific Publishing Co.
- [298] Aaron Schlutter and Andreas Vogelsang. Trace link recovery using semantic relation graphs and spreading activation. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020. Accepted: 2020-06-12T13:20:33Z.
- [299] S.S. Khan and S. Lock. Concern tracing and change impact analysis: An exploratory study. In *2009 ICSE Workshop on Aspect-Oriented Requirements Engineering and Architecture Design*, pages 44–48, May 2009.
- [300] Alexander Delater and Barbara Paech. Tracing Requirements and Source Code during Software Development: An Empirical Study. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 25–34, October 2013. ISSN: 1949-3789.
- [301] Wentao Wang, Nan Niu, Hui Liu, and Yuting Wu. Tagging in Assisted Tracing. In *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, pages 8–14, May 2015. ISSN: 2157-2194.
- [302] Senthil Karthikeyan Sundaram, Jane Huffman Hayes, Alex Dekhtyar, and E. Ashlee Holbrook. Assessing traceability of software engineering artifacts. *Requirements Engineering*, 15(3):313–335, September 2010.

- [303] Shreya Banerjee and Anirban Sarkar. Domain-specific requirements analysis framework: ontology-driven approach. *International Journal of Computers and Applications*, pages 1–25, November 2019.
- [304] Mark Grechanik, Kathryn S. McKinley, and Dewayne E. Perry. Recovering and using use-case-diagram-to-source-code traceability links. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07*, page 95, Dubrovnik, Croatia, 2007. ACM Press.
- [305] Marta S. Tabares, Ana Moreira, Raquel Anaya, Fernando Arango, and Joao Araujo. A Traceability Method for Crosscutting Concerns with Transformation Rules. In *Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design (EARLYASPECTS'07)*, pages 7–7, May 2007.
- [306] MengNi Rao and YongHua Li. Research on Semantic Judgment of Key Words in Ontology - Based Dynamic Requirements Traceability. In *2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 184–187, October 2018. ISSN: 2473-3636.
- [307] Anas Mahmoud, Nan Niu, and Songhua Xu. A semantic relatedness approach for traceability link recovery. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 183–192, June 2012. ISSN: 1092-8138.
- [308] María Luciana Roldán, Marcela Vegetti, Silvio Gonnet, Marcelo Marciszack, and Horacio Leone. An Ontology for Specifying and Tracing Requirements Engineering Artifacts and Test Artifacts. *CLEI Electronic Journal*, 22(1), April 2019.
- [309] Eman Alkhamash. Formal modelling of OWL ontologies-based requirements for the development of safe and secure smart city systems. *Soft Computing*, 24(15):11095–11108, August 2020.
- [310] Patrick Mäder, Orlena Gotel, and Ilka Philippow. Rule-Based Maintenance of Post-Requirements Traceability Relations. In *2008 16th IEEE International Requirements Engineering Conference*, pages 23–32, September 2008. ISSN: 2332-6441.
- [311] Patrick Mäder and Orlena Gotel. Towards automated traceability maintenance. *Journal of Systems and Software*, 85(10):2205–2227, October 2012.
- [312] Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grünbacher, Alex Dekhtyar, Giuliano Antoniol, and

BIBLIOGRAPHY

- Jonathan Maletic. The grand challenge of traceability (v1.0). In Jane Cleland-Huang, Orlena Gotel, and Andrea Zisman, editors, *Software and Systems Traceability*, pages 343–409. Springer London, 2012.
- [313] Mohamad Kassab, Colin Neill, and Phillip Laplante. State of practice in requirements engineering: contemporary data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014.
- [314] Stefan Wagner, Daniel Méndez Fernández, Michael Felderer, Antonio Vetrò, Marcos Kalinowski, Roel Wieringa, Dietmar Pfahl, Tayana Conte, Marie-Therese Christiansson, Desmond Greer, et al. Status quo in requirements engineering: A theory and a global family of surveys. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(2):1–48, 2019.
- [315] Jane Cleland-Huang, Orlena Gotel, Andrea Zisman, et al. *Software and systems traceability*, volume 2. Springer, 2012.
- [316] Matthias Heindl and Stefan Biffl. A case study on value-based requirements tracing. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, ESEC/FSE-13*, pages 60–69, New York, NY, USA, September 2005. Association for Computing Machinery.
- [317] Yaser Ghanam and Frank Maurer. Extreme Product Line Engineering: Managing Variability and Traceability via Executable Specifications. In *2009 Agile Conference*, pages 41–48, August 2009.
- [318] National Library of Medicine. Medical subject headings ontology, 2022.
- [319] Jesús Suaste Cherizola. From commodities to assets: Capital as power and the ontology of finance. *Review of Capital as Power*, 2(1):1–29, 2021. Place: s.l. Publisher: Forum on Capital As Power - Toward a New Cosmology of Capitalism.
- [320] Michele Missikoff, Roberto Navigli, and Paola Velardi. The usable ontology: An environment for building and assessing a domain ontology. In Ian Horrocks and James Hendler, editors, *The Semantic Web — ISWC 2002*, Lecture Notes in Computer Science, pages 39–53. Springer, 2002.
- [321] Antonio De Nicola, Michele Missikoff, and Roberto Navigli. A proposal for a unified process for ontology building: UPON. In Kim Viborg Andersen, John Debenham, and Roland Wagner, editors, *Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 655–664. Springer, 2005.

ABSTRACT

Background Requirements engineering and verification (REV) processes play essential roles in software product development. There are physical and non-physical distances between entities (actors, artifacts, and activities) in these processes. Current practices that reduce the distances, such as automated testing and alignment of document structure and tracing only partially close the above mentioned gap.

Objective The aim of this thesis is to investigate solutions w.r.t their ability to reduce the distances between requirements engineering and verification. Two techniques that are explored in this thesis are automated testing (model-based testing, MBT) and alignment of document structure and tracing (traceability).

Method The research methods used in this thesis are systematic mapping study, software requirements mining, case study, literature survey, validation study, and design science.

Results MBT and traceability are effective in reducing the distance between requirements and verification. However, both activities have some shortcoming that needs to be addressed when used for that purpose. Current MBT techniques in the context of software performance do not attain

all the goals of MBT: 1) requirements validation, 2) checking the testability of requirements, and 3) the generation of an efficient test suite. These goals are essential to reduce the distance. We developed and assessed performance requirements verification and test environment generation approach to tackle these shortcomings. Also, traceability between requirements and verification suffers from the low granularity of trace links and does not support the verification of all requirements. We propose the use of taxonomic trace links to trace and align the structure of requirements specifications and verification artifacts. The results from the validation study show that the solution is feasible in practice. However, this comes with challenges that need to be addressed.

Conclusion MBT and improved traceability reduce multiple distances between actors, artifacts, and activities in the requirements engineering and verification process. MBT is most effective in reducing the distances when the model used is built from the requirements. Traceability is essential in easing access to relevant information when needed and should not be seen as an overhead. When creating trace links, we need to consider the difference in the abstraction, structure, and time between the linked artifacts.



ISSN: 1650-2140

ISBN: 978-91-7295-442-7