CHALMERS

Automated testing of non-functional requirements based on behavioural scripts

Master of Science Thesis in the Programme Software Engineering and Technology

ZEINAB ALIZADEH BARMI
AMIR HOSSEIN EBRAHIMI

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, December 2011

Automated testing of non-functional requirements based on behavioral scripts

ZEINAB ALIZADEH BARMI
AMIR HOSSEIN EBRAHIMI

Examiner: MIROSLAW STARON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden December 2011

Acknowledgements

Next to God, we would like express our deepest gratitude to our supervisor Associate Professor Robert Feldt, Chalmers University of Technology, for his continual support and encouragements throughout the course of this thesis. His ideas and support has been our backup in every step of the project. Robert has been an inspiration to us since the first course that we had with him and all the way through this thesis, and yet regardless of how busy he has always been he has always encouraged us throughout this project. We would also like to thank Mr. Ali Shahrokni, Mr. Emil Börjesson, and Mr. Micheal Unterkalmsteiner for their comments and help in different parts of this thesis. We have learnt a lot from all of them in every step and we hope to use the lessons we have learnt in future. We would also like to thank our families for their vital support, without which we would not be in such a position. We would also like to thank all our friends for their moral support during these two years.

Preface

This Master of Science Thesis is reported here in a hybrid format, i.e. the main content of the Work is reported as 4 scientific articles. Articles “Specification of Non-functional requirements - A Systematic Literature Review”, and “Testing of Non-functional requirements - A Systematic Literature Review” conform to Requirement Engineering Journal’s template. Articles “Alignment of requirements specification and testing: A systematic mapping study”, and “Extending Behavior Driven Development for Automated Testing of Probabilistic Non-Functional Requirements” conform to International Conference on Software Testing, Verification and Validation’s template.

Table of Contents

Introduction

The first step in developing a successful software project is to properly identify and specify the software requirements in the process of Requirement Engineering (RE). This discipline aims to analyze and specify goals of the software system and also the constraints they impose on the software system. The output of this phase is a Software Requirement Specification (SRS), describing what the software system is expected to do and how it should meet the expected requirements in terms of functional requirements (FRs) and non-functional requirements (NFRs), respectively [1]. Another important branch in Software Engineering (SE) is software testing which is an essential activity in software projects since not only can it discover software faults, but it is also critical when evaluating reliability and performance.

RE and testing have a synergetic relationship with each other; requirements should specify expectations on a software system and testing should ensure these expectations are met. Moreover, both activities can directly determine the success or failure of software projects, and also their quality. Therefore, to enable high product quality and efficient development it is crucial that requirements and testing activities are aligned, i.e. testing should be as closely based on a user specification as possible. Making a strong link between them helps to discover possible errors early, which in turn will improve the product quality and lead to more satisfied customers [2]. It would also help to reach a more accurate testing plan which in turn would improve project cost and schedule estimation [3]. Alignment of requirements specification and testing is a challenging issue in SE, and although organizations are becoming more interested to establish such a link, a gap often exists between requirements and testing.

There is a broad consensus in software community that it is not adequate for a software system to meet the expected FRs without considering NFRs, as NFRs play a significant role in the success of software projects. Nevertheless, in all activities of requirements specification, testing, and their alignment the focus is mainly on FRs while little attention is given to NFRs. NFRs are typically specified briefly and inadequately and are left to be defined during architecture and design phase [4]. Their verification is almost always done very late during the development cycle, e.g. after implementation or during

integration [5]. Also, the efforts to link requirements and testing lack a proper treatment of NFRs. As the result of this underestimation, software community has experienced many software projects failure such as the famous case of the London Ambulance System in which NFRs non-compliance resulted in the deactivation of the system immediately after its deployment [6].

The goal of alignment of requirement specification and testing is to write tests as closely based on a user specification as possible. Behavior-Driven Development (BDD), as a second generation agile methodology, promotes such close alignment by basing system tests on behavioral scripts expressed closer to the terms and concepts of users and the problem domain [7]. Its specific small vocabulary facilitates communication among stakeholders and assists them to consider the software system from the same perspective by using the same words[1]. Requirements in BDD are specified as features which should have business value for the customer. Alignment of requirements specification and testing in BDD is provided through connecting textual description of requirements to test [7].

The goal of this master thesis is to summarize the existing knowledge on specification and testing of NFRs and also their alignment. We intend to provide an objective view of what studies have been done in these areas, and what type of solutions is represented in these studies. This would help to identify useful approaches and needs for future research. In order to reach this goal, we carried out two Systematic Literature Reviews (SLR) of the existing literature on NFRs specification and testing, based on the guideline proposed by Kitchenham that is suitable for software engineering researchers [8]. "A systematic literature review (often referred to as a systematic review) is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest" [8]. We also performed a systematic mapping on alignment of NFRs specification and testing which consists of an overview of primary studies and the categorization of the results by providing a visual summary [9]. It should be noted that a systematic mapping offers an overview of a field, identifying potential gaps in researches, whereas a SLR provides a more detailed study of the identified results. Another main contribution of this thesis is to investigate the possibility of aligning the specification and testing of NFRs based on BDD. The research

[1] *Behaviour-Driven Development*. Available: http://behaviour-driven.org/

question of this part is “how and to what extent BDD scripts can be used to align specification and testing of NFRs?” The question is three-fold: 1. to what extent is it possible to specify NFRs with BDD scripts? 2. which kind of NFRs can be modeled in this way? 3. is it possible to test NFR fulfillment using these scripts? In order to address these questions we propose ProBDD, an extension to BDD, in which probabilistic NFRs - such as safety, security, reliability, availability, and performance are specified in BDD scripts using NFR specification patterns from the ProProST framework [10]. ProProST is a repository of specification patterns for probabilistic properties with a structured English grammar as a textual front end. As expressions made by ProProST do not sound natural for daily use, a few alternative patterns are added to ProBDD in order to make it more applicable. The proposed extended BDD scripts can be parsed and executed with randomly generated input data and test results can verify the specified NFRs fulfillment. Since ProProST patterns are specific to NFRs that can be written as probabilistic statements, our solution is specific to NFRs that are quantified and can be expressed in a probabilistic form. However, it seems unlikely that unquantified NFRs can ever be automatically tested without human interaction and the probabilistic framework is general enough to support a large variety of NFRs.

Specification of Non-functional requirements - A Systematic Literature Review

Zeinab Alizadeh Barmi

barmi@student.chalmers.se

Amir Hossein Ebrahimi

amirho@student.chalmers.se

Robert Feldt

robert.feldt@chalmers.se

Department of Computer Science and Engineering

Chalmers University

Goteborg, Sweden

Abstract

To ensure high quality software, it is crucial that non-functional requirements (NFRs) are well specified and thoroughly tested in parallel with functional requirements (FRs). Nevertheless, in requirement specification the focus is mainly on FRs, even though NFRs have a critical role in the success of software projects. This study presents a systematic literature review of the NFR specification in order to identify the current state of the art and needs for future research. The systematic review summarizes the 51 relevant papers found and discusses them within seven major sub categories with "combination of other approaches" being the one with most prior results.

1. Introduction

The first step in a successful software project development is to properly identify and specify the software requirements in the process of Requirement Engineering (RE). RE discipline as part of software engineering process aims to analyze and specify goals of the software system to be implemented and also the constraints they impose on the software system [1]. RE includes elicitation, analysis, specification, validation, and management activities [2]. The output of this phase is a software requirement specification (SRS) that all stakeholders should agree upon [2]. This document describes what the software system is expected to do and how it should meet the expected requirements in terms of FRs and NFRs respectively [2].

There is a general agreement in software community that while it is vital for a software system to meet the expected FRs, NFRs are also important for the success of software projects. However, there are still issues which need to be addressed in dealing with NFRs. One problem is

that during requirement elicitation and specification the focus is mainly on FRs whilst little attention is given to the NFRs. NFRs are typically specified briefly and inadequately and are left to be defined during architecture and design phase [3]. The result of this underestimation is many reports on software projects failure such as the famous case of the London Ambulance System in which NFRs non-compliance resulted in the deactivation of the system immediately after its deployment [4]. Moose-test of the Mercedes Benz A Class or the Siemens mobile phone 65 series are other examples of software projects with quality problems [5]. One reason for this problem is that NFRs are often summative in nature which means they talk about general aspects of the whole system and not attributes for specific, local situations [6]. The other reasons for difficulties in dealing with NFRs could be: Firstly, they are usually subjective and abstract and it is difficult to represent them in a measurable way [7], Secondly, a formal definition or a complete list of NFRs does not exist [7], Thirdly, they often conflict with each other [8]. Moreover, it is difficult to make them visible in models since they crosscut the functional model elements [9]. Another major issue in dealing with NFRs is that there is no general agreement on how to define, elicit, classify, represent, and validate NFRs [10]. There is variety of solutions on how to document NFRs and in which place of SRS, each has some advantages and limitations [10].

In order to address the current state of the art in NFRs specification, we carried out a Systematic Literature Review (SLR) of existing literature based on the guideline proposed by Kitchenham and Charters [11] that is suitable for software engineering researchers. “A systematic literature review (often referred to as a systematic review) is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest” [11]. The purpose of this SLR is to identify, analyze, and compare the current proposals on NFRs specification. We intend to provide an objective view of what studies have been done on specification of NFRs, and what type of solutions is represented in these studies.

The rest of the paper is organized as follows: Section 2 describes the research method of our systematic review process. In section 3, the results of the review are presented and discussed in relation to the research questions. Finally the conclusion is presented in section 4.

2. Research Method

In this section we explain the design and execution of the SLR. Figure 1 shows the steps of the systematic review that will be described in detail in the following subsections.

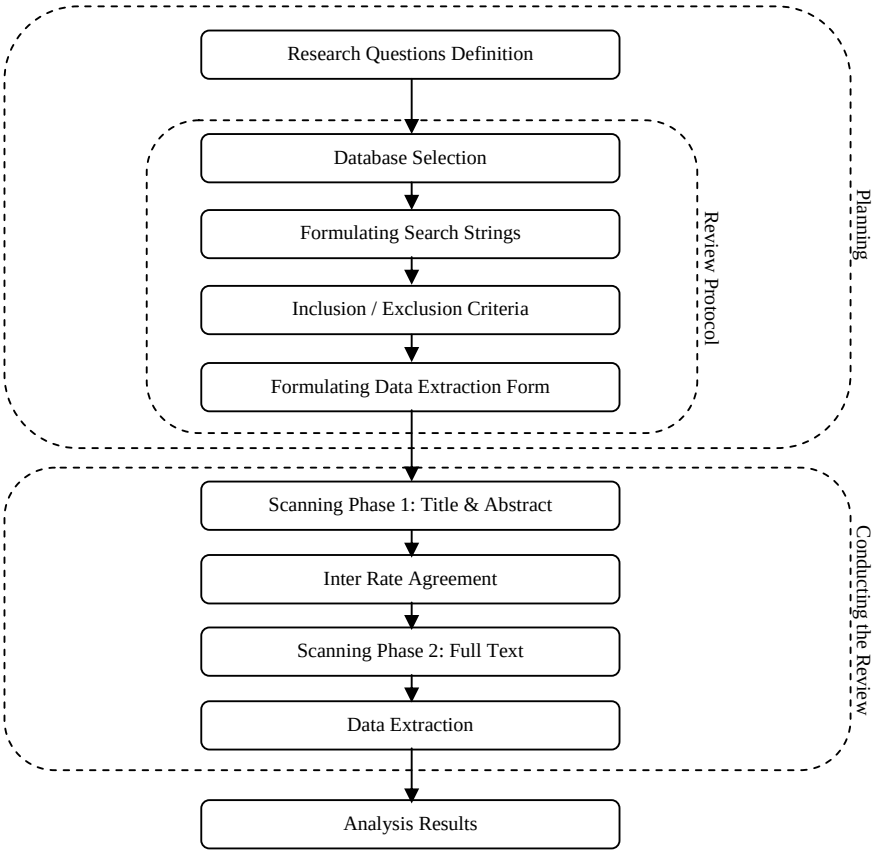

Figure 1. Systematic Review Steps

2.1 Definition of research questions

Research questions are the most important and pivotal part of any systematic review. Once they are precisely defined, a search strategy can be devised for the systematic review to identify relevant studies and extract the data required to answer the questions [12]. The main focus of this paper is on NFRs specification research. In order to carry out this investigation we defined the following research questions:

ID	Question	Aim
RQ1	Which studies have been done on Specification of NFRs?	We want to find out which topics have been investigated and to what extent? What are the different perspectives that address the specification of NFRs? E.g. use case driven approaches, goal- oriented approaches, etc.
RQ2	What type of solutions is represented in the research on NFR specification?	We want to find the contribution type given in different topics that address specification (such as method, process, etc) and also research type, e.g. solution proposal. We also aim to know the extent with which the validity of these solutions has been discussed.

Table 1. Research questions

2.2 Developing Review Protocol

In this section we will explain our review protocol. Review protocol aims to decrease the potential researchers' bias [11] in systematic reviews. It also makes it possible to replicate the review in the future [11]. The preliminary protocol was revised, evaluated, and improved by another author with expertise in conducting systematic reviews. The following sections summarize the protocol. There are four main steps in a review protocol:

1. Selecting the source databases
2. Formulating the search strings
3. Definition of inclusion-exclusion criteria
4. Formulating data extraction forms

2.2.1 Database selection

The first step in the process of paper selection is to choose appropriate databases on which our search queries are going to be executed, which will result in a set of selected papers. As this set of papers is used for all the inclusion and exclusion in the systematic review, choosing the appropriate database and search query carefully is of utmost importance, and will directly affect the completeness of the systematic review. The chosen databases were: *IEEE Xplore Digital Library, SciVerse Scopus, Inspec Engineering Village, ISI Web of knowledge, and ACM Digital Library*. *IEEE Xplore Digital Library and ACM Digital Library* cover almost all important conferences, workshops, and journal papers, which are published either by IEEE or ACM. *SciVerse Scopus* claims to be the largest database of peer-reviewed literature and quality web sources. *Inspec Engineering Village* and *ISI Web of knowledge* are also globally accepted source of choice for software researches. In order to handle the high number of found publications, we decided to use a reference management system. We chose EndNote X3[1] that is an industry standard software tool for storing, managing, and searching bibliographic references. Later on, the last database (ACM Digital Library) was removed from the list due to its lack of support for exporting search results in batch into EndNote X3.

2.2.2 Formulating Search String

In order to specify our search string we needed to gain an overview of NFRs specification. So we gathered an initial set of previously known publications in the area, through an exploratory search [13-14,2,15-16,6,17-18,9,19]. We then tried to extend our initial set using forward/backward referencing, i.e. looking at which papers were referenced in or referred to papers in our initial set. The search string was formulated according to the following steps:

1. Identifying some major search keywords using the research questions
2. Finding alternative and synonyms for these search keywords
3. Grouping the search keywords into two categories. These categories focused on NFRs (named C1), and specification of NFRs (named C2).

[1] http://www.endnote.com/enabout.asp

4. Refining the search string, as a result of application of this query to the chosen databases, it was observed that the number of hits was very high, and so was the number of irrelevant papers, e.g. papers from other fields of engineering except software engineering. Hence, constraints were applied to the search string, e.g. papers only written in English, only papers published in the last 10 years (between 2000 and 2010), etc. It should be noted that since the search was done in November of 2010 not all 2010 papers will be included in the results. It is worth mentioning that the overall search string was the result of an incremental refinement that was carried out in five steps. In each step the results were checked to see if they contained papers from the initial set on the NFRs specification. In case the items of the initial set were missed or the results were not relevant the search string was improved further. The final set of keywords which constructed the search string is shown in Table 2. The two categories are joined together using an AND Boolean operator.

NFR – C1
"Nonfunctional requirement" OR "Nonfunctional requirements" OR "non functional requirement" OR "non functional requirements" OR "nonfunctional software requirement" OR "nonfunctional software requirements" OR "non functional software requirement" OR "non functional software requirements" OR "nonfunctional property" OR "nonfunctional properties" OR "non functional property" OR "non functional properties" OR "quality attribute" OR "quality attributes" OR "quality requirement" OR "quality requirements" OR "quality attribute requirement" OR "quality attribute requirements"
Specification – C2
"specify" OR "specifying" OR "specification" OR "represent" OR "representing" OR "representation" OR "document" OR "documenting" OR "documentation" OR "formulate" OR "formulating" OR "formulation" OR "express" OR "expressing"

Table 2. Search string categories

Finally, the search was performed on the chosen databases resulting in 769 papers**.**

2.2.3 Inclusion-exclusion criteria

Inclusion/exclusion criteria are used to reach a common understanding between team members on the conditions under which a paper should be included or excluded. The main criterion for inclusion was papers focused on NFRs specification. Another important criterion for inclusion of papers was if they had been subject to peer review. Papers which were on the definition, classification, or categorization of NFRs should be excluded. Also papers not focused on software development, e.g. papers that focused on hardware or network development, posters, conference proceedings, short papers (with less than 6 pages), and opinion papers i.e. papers that express the personal opinion of author on what is good or bad [20] should also be excluded.

2.2.4 Formulating Data Extraction Form

During this phase a data extraction form was developed, partly based on the form used by Ivarsson and Gorschek [21]. Some keywords and concepts, like context and domain were reached through the study of paper abstracts by each researcher.

2.3 Selecting Studies

In the first step each member of the team scanned the title and abstract (if needed) of the papers, based on the inclusion-exclusion criteria. Papers which the team was uncertain whether to exclude or include, were left for the second step of scanning. After the first step of scanning 74 out of the 769 papers were included. During the scanning of the abstract some key attributes of the initial data extraction form were evolved. The final set of key attributes, their values and description are shown in Table 3.

Key attributes	Values	Description
Research focus	Extension of FRs specification approaches (UML profile, Use cases), NFR Specific (Goal-Oriented, Aspect oriented, etc.), Combination of other approaches	Studies are categorized in different research focus groups in order to show how different software development approaches address NFRs specification.
Contribution type	Tool, process, model, framework, guideline, method, metric and other	Specify the contribution type and other required details like name of the tool, framework, etc.
Focused quality requirement or attribute	From the reviewed literature, general	Whether the research is focused on one specific quality requirement or is it general for all quality requirements
Research type	validation research, evaluation research, solution proposals, conceptual proposals, opinion papers, and experience papers	The classification of research type is based on the classification proposed in [20].
Context	academia, industry, and open-source software	Whether the research is performed in academia, or is it the result of an experience in industry [21].
Domain	real-time systems, safety-critical systems, web systems, information systems, process controlled systems, banking and finance, education, energy resources, government and	If the research has been done for a specific type of application or domain then this key attribute is used to state that domain, otherwise it will be filled with general. The values for this key property are adopted using a well-known application domain taxonomy

	military, insurance, medical/health care, telecommunication services, transportation, other	from Digital's Industry Taxonomy [22].
Scale	Toy example, Down-scaled real example, Industrial, N/A	This key attribute captures the scale of application which the technology was used on in the evaluation. The values of this key attribute range from toy examples used in student projects and small examples to industrial scale applications [21].
Benefits and drawbacks of using this method	From the reviewed literature, N/A	This key attribute captures benefits and drawbacks of the proposed approach which are mentioned in the research
Perceived industrial benefits of this method	From the reviewed literature, N/A	This key attribute captures benefits offered by the proposed approach to the industry, as mentioned in the research
Context described	Weak, medium, strong	If there is no specific context provided for the study the value of this key attribute would be weak. The medium value is given when the organization / company and development efforts for the study is briefly described. Otherwise, in case of a detailed description for the organization / company and development efforts the strong value will be given to this key attribute [21].
Evaluation discussed	Weak, medium, strong	This key attribute shows the degree to which the evaluation of the study is discussed. The evaluation value is weak if the paper does not discuss the evaluation of the study. If it has been mentioned in the paper that the solution is evaluated but not in any detail the value will be medium. Otherwise if the paper has discussed the evaluation then the value will be strong [21].

Table 3. Key attributes of data extraction form

Before proceeding to the analysis of the review, an inter rate agreement was defined to ensure that all team members have the same understanding of inclusion exclusion criteria. The inter rate agreement was calculated based on each team member result for randomly selected

subset of all the papers, 10 %, using ReCal online utility [23]. The calculated inter rate agreement (96.1%) was at an acceptable level.

In the second step, the full text of the included papers was studied during which data extraction forms were filled for each paper. If researchers did not agree on the inclusion or exclusion of some papers they were discussed until a decision was made by consensus. At the end of this step of scanning 25 papers remained included in for the NFRs specification. We also decided to perform a snowball sampling [24] in combination with our systematic review, so that the relevant publications in the last 10 years would be used to find earlier relevant papers. As a result of snowball sampling another 26 papers were found, two of them were published before 2000. The rest were published in the last ten years, but were not found by our search string. The reason is that their title/abstract only included "requirement" keyword. However, they did consider NFRs in their proposed solutions. Totally 51 papers were included for scanning.

2.4 Validity Threats

The main threats to validity of the systematic review are publication and selection bias, and data extraction, each detailed below:

2.4.1 Publication and selection bias

One of the threats to the validity is the process of paper selection.

- The search string might miss some papers relevant to the review. Although our search string seems logical to find papers on NFRs specification, a more detailed look at the results concerning the papers in the initial set showed that some well-known relevant publications did not have any term included in our C2 (specification) category in their title or abstract. A good example is the "*On Non-Functional Requirements in Software Engineering*" paper by Chung et al [25]. To avoid missing such papers and having a comprehensive research, a reasonable solution was to combine our systematic review with snowball sampling.
- Inclusion of papers based on the information provided in the title/abstract might miss relevant papers. However, if NFRs specification is not mentioned in the title/abstract of a paper, it is more likely that this issue is not the main contribution of the paper; therefore this threat will be limited.
- The decision for inclusion/exclusion of papers is subjective. Although paper selection process was carried out on the basis of criteria to minimize such a threat, the use of these criteria could still prove subjective. Three steps were taken to limit this potential threat. Firstly, a pilot paper selection (on 100 papers) was performed before the actual process, based on inclusion/exclusion criteria, in which team members aimed to reach a common understanding of the inclusion criteria. Secondly, title/abstract scanning was performed by all team members in parallel and the outcome was cross-checked. In case of disagreement those specific papers were discussed, if a mutual result was reached the disagreement would be resolved otherwise this paper was left for the detailed study in the

next step. Thirdly, after the first step of the scanning process an inter rate agreement was calculated to ensure that as a group, all team members had an acceptable level of understanding of the selection criteria.

- Relevant papers that are published before our publication year constraint will be missed. After investigation, this threat was also found to be limited as the majority of the research before this period was focused on FRs. Additionally, performing a snowball sampling helps to decrease this threat.

2.4.2 Data extraction

Assignment of values to different key attributes of the data extraction form could be subjective. In order to mitigate this threat each team member analyzed the selected papers of the pilot paper selection and filled the data extraction forms. Then the resulted data extraction forms were compared to find the necessary modifications of values/descriptions of key attributes, and also to reach a common understanding among team members.

3. Results and Analysis

The following sections outline the results and analysis for each research question.

3.1 RQ1. Which studies have been done on Specification of NFRs?

The results of the SLR are divided into three main categories according to their respective research focus. Studies in the first category extend approaches which have been mainly used for FR specification. These approaches are UML profile and Use cases. The second category contains approaches which do not extend FR approaches in order to specify NFRs. Instead, they either focus on NFR specification e.g. patterns or propose solutions that can be used for both FRs and NFRs e.g. goal-oriented. Studies in the third category propose solutions by integrating and combining different approaches from the first and second groups. Table 4 shows the studies classified by the research focus. Distribution of publications' research focus is shown in Figure 2. Furthermore, the distribution of publications over years is shown in Figure 3. Benefits and drawbacks of research focuses are discussed at the end of each corresponding section. Benefits are categorized based on several criteria including: improving NFRs specification, ease of use, reusability, and contribution to other SE process steps. Improving NFRs specification can be determined by some factors like a decrease in the level of ambiguity or an increase in the level of simplicity, understandability, and completeness. Some factors which determine an approach's ease of use include providing a guideline or process to support NFRs specification. Contribution to other SE process steps shows the possible effects of the specification approach on other SE phases like design, implementation, etc. Reusability answers the question that whether any part of the specification approach, like artifacts, is able to be reused in other SE phases or other solutions? It should be noted that the mentioned benefits and drawbacks are specifically stated in these papers and are not the result of our interpretation.

Research Focus		Papers
Extension of FRs specification approaches	UML profile	[8], [26-27]
	Use cases	[3], [16,28-29], [30], [31], [32], [33], [34], [35]
NFR Specific	Goal-Oriented	[36], [37], [38], [39], [40], [41], [42], [43], [17], [44]
	Aspect Oriented	[45], [46], [47], [48]
	Quality Model	[2], [5], [15], [34,49], [50]
	Patterns and Languages	[6], [14], [51], [52], [53], [54], [55]
Combination of approaches	[39], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65]	

Table 4. Publications classified by the research focus - NFRs specification

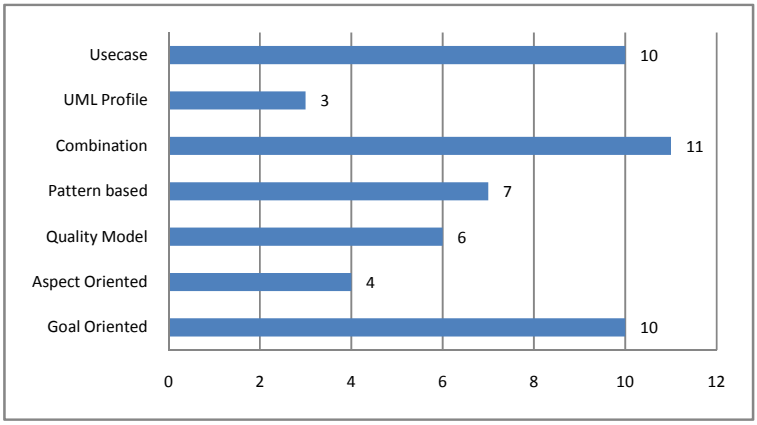

Figure 2. The research focus distribution of the publications - NFRs specification

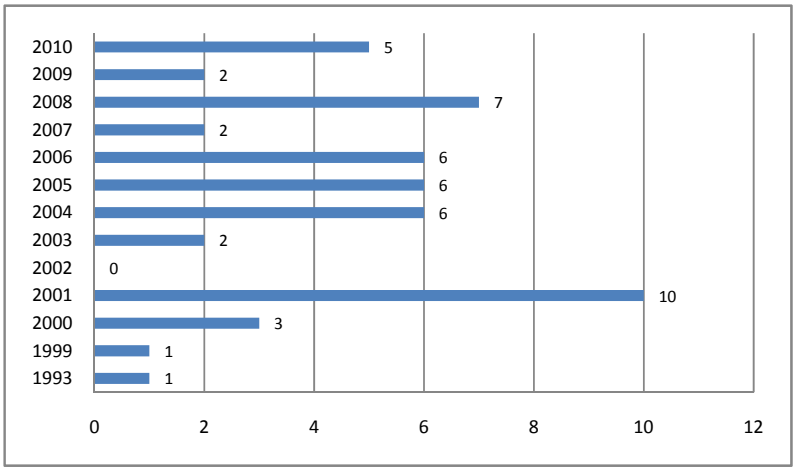

Figure 3. The distribution of publications on NFRs specification according to year

3.1.1 Extension of FRs specification approaches

3.1.1.1 UML profile

UML profile is proposed in some prior research in order to specify NFRs in UML. UML profile is a collection of UML notations to specialize UML for a particular domain. For instance, Hussein and Zulkernine propose a framework [8] to specify security requirements, particularly intrusion scenarios. Intrusion scenarios explain the steps of an illegitimate access to some protected resources of the system in abstract level. The framework provides a UML profile called UMLintr (UML for intrusion specifications) that extends UML notations with tagged values, stereotypes, and constraints. This extension solves the problem of attaching many notes to UML notations in order to specify NFRs. UML artifacts for documenting the intrusion scenarios in the framework are use cases, class diagrams and state-machines. Another UML profile is UMLsec [26-27] in which stereotypes and tagged values – such as integrity, and authenticity are used to add security requirements to UML diagrams. It also supports the interpretation of these tagged values using Object Constraint Language (OCL). UMLsec differs from UMLintr as it concentrates on specifying and validating security requirements rather than intrusion scenarios.

Benefits

Improving requirement specification

UMLintr helps to avoid redundant, conflicting, or ambiguous requirements [8].Communication with stakeholders using extended UML diagrams in UMLintr is also much easier than using attack language [8].

Ease of use

UMLintr simplifies intrusion scenarios specification as developers are not required to learn a separate language for describing attacks [8]. Similarly, developers are not required to be security experts in order to use UMLsec, as UML is used to encapsulate knowledge on prudent security engineering in this approach [26].

Drawbacks

A drawback of UMLintr is that it does not support specifying distributed attacks in its UML profile. As a result the profile has to be changed accordingly, whenever an intrusion that cannot be specified using UMLintr is found [8].

3.1.1.2 Use cases

Use case driven approach to software development, promoted by the UML, is a well known mechanism for requirement specification [28]. While the focus of this approach has been primarily on FRs, recent approaches also encompass specification of NFRs. Sindhgatta and Thonse motivate the application of use cases for NFRs specification by outlining the problems of textual description and formal methods as two extremes for requirement specification [16]. Using textual description is simple but can cause ambiguity; on the other hand, formal specifications are precise but difficult to understand and use by practitioners. Sindhgatta and Thonse believe that use case is a concept between these two extremes and could be effectively applied for requirement specification if it is

used carefully. For instance, if requirement analyst put a lot of description in use cases, understanding and verifying them would be difficult for stakeholders. They propose the Use Case Specification Framework with the aim of making use case diagrams simpler and more structured by using Business Process Modeling (BPM) notations in uses cases [16]. The framework is able to capture performance and usability requirements and map them along with the use cases. To reach these goals, they adapt the process workflow notations of BPM for representing use case flow of events. Laibinis and Troubitsyna propose a general pattern for use-case modeling of safety- critical systems in different levels of abstraction [30]. A fault tolerance requirement is an ability of a system to cope with errors. Use cases in this approach are structured in order to capture their fault tolerance requirements. To do so, an auxiliary use case which describes the actions for error recovery in case of use case failure should be added to every use case. Daniels and Bahill provide a hybrid approach in which "shall statements" are used within UML diagrams to effectively express both FRs and NFRs [31]. They mention that use cases are understandable at the expense of being potentially ambiguous. On the other hand, "shall statements" are very well defined but stand alone. Specifying requirements as a series of discrete and disembodied "shall statements" without context makes it difficult for the human mind to fully understand the intent and dependencies of each and every requirement, which in turn will make it hard to detect redundancies and inconsistencies. This tension promotes the use of a hybrid approach in which both UML and "shall statements" are used. This approach proposes to use "shall statements" in the special requirement section within use case reports.

In order to specify NFRs in this approach, some prior research propose to modify use case structure. For instance, Adolph et al. discuss that including NFRs in the description part of FRs will make use cases distracting, and harder to understand [29]. They suggest adding some fields in order to capture NFRs in use cases [29]. Schneider and Winters also suggest to add a special requirements section to the use case [28].

Other solutions for specification of NFRs in use cases propose some complementary form of use cases, e.g. misuse case and control case. Misuse cases can be viewed as negative form of use case by predicting the possible misuses of the software system [32]. They also help to elicit the appropriate solutions to prevent or mitigate misuses by deriving new requirements or even new subsystems for handling the exception events caused by threats [32]. Misuse case can be efficiently used to specify NFRs whose goal is to handle software threats. One such NFR is security requirements as they are written in order to prevent or mitigate threats of negative agents' activities which want to misuse the system or even cause failure [32]. To specify security requirements, analyzer identifies the threats at a high level of abstraction, and specifies the use cases for preventing or mitigating them. Each of these use cases might be threaten by negative agents, so other misuse cases can be added in this way [32]. Developing misuse cases and use cases can be done recursively, going from system to subsystem levels, or lower [32]. Sindre and Opdah also apply misuse cases for security requirements specification [33]. They mention that textual description of use cases or misuse cases are more important than their diagram representation. Based on this idea, they provide templates for misuse cases that capture security requirements. These templates are adaptations of templates being used for use cases such as the

one suggested by Kulak and Guiney [66] or the template suggested by Cockburn [67]. Although misuse cases are mostly used for representing security requirements, there are other NFRs which can be specified by them. Alexander proposes solutions to specify safety, reliability, usability, maintainability, portability, and hardware aspects like storability and transportability in this way [32]. One of the issues that should be addressed in applying misuse case is that they are usually used intuitively and there are few systematic methods for deriving them [34]. One such method is misuse-oriented quality requirements engineering (MOQARE) in which quality requirements can be derived based on a general conceptual model and in a checklist-based process in a top down fashion [35]. The conceptual model is based on these elements: business goals, quality goals, quality deficiency, threat, misuser, vulnerability, misuse case, and countermeasure. A misuse case documents the reason, how and by whom a "quality deficiency" is caused. "Countermeasure" counteracts the "vulnerability", "threat", "misuser", and negative consequences. The derived elements are presented in a hierarchical graphical graph called ''Misuse Tree''.

Control case is a useful technique to specify NFRs which cannot be addressed by use cases [3]. So, it can complement use cases to provide a more complete view of the system requirements. Control cases have two elements:

1. "Operating condition" which models the constraints. The operating condition can be associated with a use case in requirement specification phase. In later phases it can be associated with other elements such as a process activity or UML activity diagrams.
2. "Control case" which specifies a set of quality statements that the system must meet in order to manage the risk identified by the operating condition.

The "control case" and "operating condition" were introduced in [68] and further refined in [69]. Control cases can be applied during the software development process [3]. When use cases are specified for a system, the operating conditions and control cases are also identified and specified in an abstract level. After that, when use cases are elaborated to capture more details of functional requirements, control cases are also elaborated to capture the detailed NFRs.

Benefits

Improving requirement specification

- Simplicity of requirements specification is improved in use case modeling through providing a semi-formal structure [29]. This approach also facilitates to structure complex requirements [30]. Misuse cases also improve simplicity [35,32] and requirement organization [33]. NFRs which can be specified with Misuse case, i.e. NFRs whose goal is to handle software threats are handled in a uniform way as all of them state what should not happen [33].
- Understandability of requirements among stakeholders is improved by using control cases which in turn facilitate their communication and collaboration. This would allow easier discussion and agreement on quality requirements among stakeholders [3]. Improvement of understandability is addressed in [31], using "shall statements", and in [35,32] through use of misuse cases.

Contribution to other SE process steps

- Use cases are helpful in early detection of software errors since they describe system requirements for error situations at different levels of abstraction [29].
- Misuse cases can contribute to NFR elicitation and validation [32]. They can also be applied efficiently to requirement analysis, design trade-offs, and verification. As misuse cases show the rationale of using a certain defense, they can help later in the maintenance phase [33]. Context-rich misuse case scenarios, checklists, and the possibility to choose the order of NFRs elicitation allow an intuitive and creative elicitation in MOQARE [34]. In addition to software requirements elicitation, MOQARE enables discovering requirements and constraints on the software development process or the project [35].
- Control cases help in early discovery of NFRs and accurate capture of requirements [3]. Control cases can also help developer in implementation phase to make correct coding decisions by reminding him that there are acceptance criteria in control case to be met. During testing phase tester should simulate "operating conditions" to verify whether the system meets the requirements specified in the control cases.
- The proposed pattern in [30] enforces early consideration of fault tolerance requirements and also supports reasoning about them. The pattern is used as a base for system design validation. It also improves requirement traceability by making requirements engineering process more structured [30].

Ease of use

- Use cases are easy to use due to their semi-formal structure, even for users with very little training [29].
- Misuse cases are helpful in making design decisions, eliciting test cases and validating requirements.
- MOQARE supports intuitive and systematic specification of NFRs using clearly defined concepts which are supported by a notation with tree structure [35,34]. The hierarchical graphical structure of misuse tree helps to structure interviews and form of results [35,34]. Checklists help to avoid focusing on only a few quality attributes, threats, or misusers [35]. In order to address ease of use, [33] also provides guideline on describing use of proposed template in detail.

Drawbacks

- Representing NFRs in use cases is not straightforward since sometimes NFRs do not affect a single FR or the system as a whole; instead they affect a sub-set of FRs that might described in more than one use case [19]. Some Studies use a special part of use case description to describe NFRs [28-29]. This kind of representation could have some disadvantages which include reference points for some NFRs becoming non-intuitive, low level of organization for NFRs due to lack of modeling constructs, lack of traceability between NFRs and other software artifacts, and possible source of redundancy and error proneness when NFR textual description is duplicated in many use cases.

- MOQARE also has some drawbacks. Translating informal requirements into misuse tree requires a method specialist [35]. Items in checklists might be too general which makes the requirement analysis difficult as it aims to find system specific requirements [35].

3.1.2 NFR specific approaches

3.1.2.1 Goal-Oriented approaches

Goal-oriented requirements engineering is concerned with the use of goals for eliciting, specifying, analyzing, and modifying requirements. The system which is under construction should achieve an objective named the goal [36]. Goals could have different levels of abstraction, from the high-level, strategic concerns to low-level, technical concern. One of the accepted taxonomies for goals, is functional and non-functional goals [36]. Services that a software has to deliver are represented by functional goals; whereas non-functional goals refer to quality attributes which needs to be addressed while delivering those services [36]. Another distinction which is often made in the literature are hard and soft goals [36]. Functional goals are classified as hard goals, as their satisfaction is objective and can be established and verified using (formal) verification techniques [36]. On the other hand non-functional goals which are represented by softgoals cannot be established in a clear cut sense [36]. As a result of these taxonomies we would have four groups of goals which are: functional hardgoals, (objective goals about the services which the software has to deliver), non-functional hardgoals (objective criteria for how the service is to be delivered), functional softgoals (imprecisely software services), and non-functional softgoals (imprecise statements for how a service is to be delivered). There is a big difference between a requirement and a goal: A goal might need cooperation of a hybrid of active components to be achieved; a requirement from this point of view is a goal under responsibility of a single active component in the system-to-be. One main problem of many goal-oriented approaches is the lack of clear distinction between goals and requirements. This problem is highlighted by the question “when to stop refinement” [37]. One proposed solution is to proceed with goal refinement until the goals are assignable to agents (i.e. human, devices, and software) [38]. Another solution is to stop refining when goals don’t refer to domain properties, which naturally exist in the environment [70].

One of the main problems of software projects is that elicited requirements, goals, and assumptions are sometimes too ideal in the early stages [38]. This over-idealization might result in unrealistic, unachievable and incomplete requirements [38], which in turn might increase the chance of software failure. One solution proposed by Lamsweerde and Letier is based on obstacles [38], which are a dual notion to goals; while goals are all about desired conditions, obstacles capture undesired ones. Obstacles can obstruct some goals. This research gives techniques for formalization of obstacles and their systematic generation, and also a set of alternative operators for goal specification transformation in order to resolve the obstacles which were generated. Plosch et al. suggest a bipartite approach to elicit and specify quality requirements based on goals and operational quality models [37]. In order to overcome the problem of systematically refining goals and specifying when to stop the refinement process, they combine obstacles with factors [37]. This approach will help to identify more quality requirements. Also, the newly identified

quality requirements may help to enhance existing quality models by further quality aspects and measures.

The NFR framework is a well-known goal-oriented approach for addressing NFRs [39-40]. NFRs in this framework are represented by softgoals which need to be satisfied. When there is enough positive and little negative evidence for their claim softgoals are said to be satisfied [39]. In this framework, operationalizing softgoals is the process to represent design decisions for achieving NFR softgoals. In this process softgoals are identified, analyzed, and recorded in Softgoal Interdependency Graph (SIG). In SIG, NFR softgoals are specified as Type (Topic) where Type is a non-functional aspect (e.g. serviceability, performance) and Topic is the context for the non-functional aspect (e.g. pricing system).Type or Topic is used for the refinement of NFR softgoals, using AND/OR decomposition.

In Knowledge Acquisition in automated specification (KAOS) [36], another pioneer goal-oriented method, goals are considered as a set of high level constraints over states. They are distinguished by type (such as functional/non functional, hard/soft goal), attribute (such as their name and specification), and their links. Links could be between goals or between goals and other elements of requirements models. Goals are formalized through keyword verbs such as Achieve, Maintain and Avoid. They mean that a corresponding target condition will be eventually satisfied in the future, will be satisfied in future unless some other condition holds, and will never be satisfied respectively. KAOS does not differentiate between functional and non-functional goals, but the differentiation could be represented by the KAOS graphical AND/OR graph [25]. Dardenne et al. propose a conceptual meta-model in the context of the KAOS project for specification of system and its environment [44]. The meta-model is represented by a graph with each node capturing an abstraction such as goal or agent, while the edges capture semantic links between these abstractions. Requirement models are then instantiated from this conceptual meta-model. The strategy for instantiating requirement model is also provided in the study [44].

Tropos is another software development methodology with the aim to reduce the mismatch between software system and its operational environment [41]. In Tropos, information system is viewed as one or more actors who participate in the fulfillment of functional and non-functional stakeholder goals [41]. It adopts the concepts offered by i* such as actors and social dependencies among them including goal, softgoal, task and resource dependencies for modeling early requirements [41]. There are four phases in Tropos: Early requirements, late requirements, architectural design, and detailed design [41]. In the first phase, the focus is on understanding the system-to-be, the stakeholders' needs, and modeling them as goals by goal-oriented analysis. This results to documenting involved actors, their respective functional and non-functional goals and their inter-dependencies in an organizational model. In the second step, all FRs and NFRs of the system-to-be is described within its operational environment. Adopting i* concepts provides an informal graphical representation of the organizational setting. Fuxman et al. present a framework based on Formal Tropos (FT) specification language [42] that supplements adopted i* concepts with a rich temporal specification language inspired by KAOS [36]. Reaching satisfactory formal specification based on i* models is not an easy task. For decreasing this difficulty, one needs to extract as much information as possible from i* model while conducting an initial FT model.

Secure Tropos as an extension of Tropos provides modeling and analysis of security requirements through integrating trust, security and system engineering [43]. Secure Tropos has been chosen by Onabajo and Jahnke [17] in their approach to precisely specify confidentiality requirements. The approach uses properties of confidentiality requirements in order to formally represent them. The identification of these properties is systematically performed using data sources such as stakeholders concern on confidentiality, and confidentiality legislations and policies.

Softgoals intuitiveness and ease of use has led to their integration in Goal-Oriented RE (GORE) frameworks like, i*, Tropos, Goal-oriented Requirement Language (GRL), and Requirement Engineering Framework (REF) [40]. But still one of the disadvantages of softgoals remains the fact that they are informally specified and used [40]. Jureta et al. help to reach a deeper understanding of requirements which are expressed in goal diagrams as instances of the softgoal concept [40]. Softgoal concept includes information which is mainly subjective, context-specific, and imprecise and contains preferences of the stakeholder. Jureta et al. present an extended characterization of the common softgoal concept for representing and reasoning about quality requirements in the early stages of requirement engineering [40]. They also suggest guidelines for use in the requirement modeling approaches which aim to employ the given softgoal conceptualization.

Benefits

Improving requirement specification

- Addition of different information to the description of softgoals in [40] allows NFR specification to be more precise, less subjective, less contexts specific and less idealistic. Imprecision is decreased in this approach through the clearer identification of its sources [40].
- Obstacle based approach in [38] would result in a more realistic, complete and robust requirement specification.
- Similarly, inadequacies, incompleteness and contradictions in requirements specifications would be easy to find by matching of parts of the meta-model to the requirements in [44].
- In [17], Secure Tropos is used to make a more precise and complete confidentiality requirements.

Contribution to other SE process steps

- In goal oriented approaches, goals show the rational of requirements [36]. Additionally, traceability from high level goals to low level technical details is provided using goal graphs [36].
- Tropos reduces the gap between software systems and their operational environment [41]. Tropos is also successful in detecting critical bugs which are not easy to find in informal settings [42].
- Formal Tropos supports formal analysis and verification of early requirements specifications [42].

- The NFR framework can be adopted easily into the elicitation and analysis methods [57]. This framework also aids with the analysis of unclear NFRs.
- The proposed approach in [17] for confidentially requirements specification allows automated analysis of the elicited confidentiality requirements by representing them in a formal way.
- Use of quality models in [37] has resulted in requirements with clear criteria for testing. Additionally systematic collection of quality requirements with an emphasis on providing fit criteria can help in reaching a higher level of quality assurance and user acceptance.

Reusability

- Specification of quality requirements can be shared and reused as it is based on quality models in the approach proposed in [37].
- Tactics and conceptual models given in [44] can also be reused in other solutions.

Ease of use

- The NFR framework is quite simple, intuitive and easy to understand [57].
- In order to employ the proposed softgoal concept extension in [40], users are provided with a guideline to transform softgoals to the new format.

Drawbacks

- Deciding when to stop goals or obstacles refinement is challenging in goal oriented approaches due to the lack of distinction between goals and requirements [37-38] .
- In the approach proposed in [37], quality requirements which are not mentioned in the quality model might be overlooked.

3.1.2.2 Aspect-oriented approaches

In Aspect-Oriented Software development (AOSD), a concern is a property or an interest point of a system [56] and an aspect is a modular unit designed to implement a concern [71]. A crosscutting concern is a concern which can be found across multiple modules or entangled in a unique module like data persistency, transaction security, or error handling [45]. Crosscutting concerns cause two main problems for software development due to their very nature [45]. Firstly, their design and implementation is scattered over many building blocks (the scattering problem). Secondly, one building block often comprises the design or implementation of more than one concern (the tangling problem) [45]. AOSD aims to overcome these problems by modularizing crosscutting concerns and encapsulating those using aspects [45]. The decision to separate concerns at implementation level is very late and costly, as many of the project decisions have already been taken [72-73]. To deal with this issue, many approaches were introduced to handle identification, separation and composition of concerns early in the life cycle of software, beginning at the requirement engineering stage [72-73].

Amirat et al. adapt use cases in aspect oriented development approach [46] in order to early discovery of crosscutting FRs and NFRs starting from requirement elicitation. The crosscutting requirements are integrated with appropriate requirements using the mechanism provided by this approach. The process steps are as follow: In the first step, FRs are identified and

documented by use case diagrams at high level of abstraction. Then they are described in more detail using textual description, graphical representation, and formal specification. If a use case is repeated more than once in the system functional description, then it is a crosscutting. In the second step, NFRs are identified using NFR catalogue [13] in which each item of the category is checked if it is applicable in the system. Identified NFRs are represented through a matrix which relates NFRs to their affected use cases. In the third step, the interaction points -the places of the system which are affected by crosscutting requirements are identified; conflicts among requirements in these interaction points are identified and resolved; and finally requirements are integrated.

Some aspect-oriented approaches address special domains such as From RequIrements to Design using Aspects (FRIDA) [47]. FRIDA is a method which offers a sequence of phases to support requirement analysis and system design in Distributed Real-time Embedded environments (DRE). Dealing with the complexity of NFRs separately from FRs is the main goal of this method [47]. FRIDA is divided into six main phases [47]. In the first phase, FRs are identified and specified via use case diagrams and other templates. In the second phase, NFRs are identified and specified using checklists and lexicons. Conflicts among the NFRs should be resolved in this phase. Classes, actors and use cases are linked in the third phase. This linking is done for the NFRs in the next phase. In the fifth phase FRs are represented as classes which are linked to aspects. In the last phase the source code for the classes and aspects is generated.

Bombonatti and Melnikoff have carried out a survey [48] on comparing several early aspect detection methods. Their comparison criteria are: concern elicitation, identification, representation, and composition [48]. In this paper we will focus on the concern representation part and methods which deal with NFRs specification.

- AORE with UML: This method is a UML based proposal based on the general Aspect-Oriented Requirement Engineering (AORE) process in order to separate and compose crosscutting concerns [48]. NFR concerns are identified as a restriction on the system which has to be satisfied. NFR crosscutting concerns are identified as concerns which affect more than one use case, and they are described in specification templates. There is no clear guideline for identification of NFR crosscutting concerns.
- Aspectual Use Case Driven approach: In this approach, crosscutting concerns are identified if they constrain, extend or are included by more than one use case. Non-functional concerns are described in specification templates and functional concerns are described with use case models [48]. There is no clear guideline for identification of non-functional concerns.
- NFR framework integrated in requirement engineering model: Integration of NFR framework in requirement engineering model is proposed based on “Aspectual use case driven approach”, differing in the composition activity [48]. Most appropriate methods are used for functional and non-functional concern identification, which could include use cases, viewpoints, or NFR frameworks. Crosscutting concerns are the ones which are requested by more than one concern. Concerns are represented by specification templates, use case models and SIG diagrams. Non-functional concerns identification is vague and abstract in this method.

- AORE integrated with NFR framework: With an initial focus on non-functional crosscutting concerns separation and composition, AORE is integrated with NFR framework [48]. Non-functional crosscutting concerns are represented as softgoals in SIG diagrams and functional crosscutting concerns are represented by use case models.
- Concern-Oriented Requirements Engineering Model: Concern-oriented requirement engineering is proposed based on AORE [48]. Identification of concerns is done using techniques like use cases, viewpoints and goals. Concerns which affect several other concerns are classified as crosscutting concerns. The specification is represented by the XML language with some predefined tags.
- Aspect-Oriented Development Model with Traceability Mechanisms: This method aims to present an approach to complement the separation and composition of crosscutting concerns with traceability mechanisms for dynamic and static views [48]. Use cases and scenarios are traced via dynamic views and conceptual classes are traced via static views. Use cases which are used to specify functional concerns are classified as crosscutting if they are included by several use cases. Non-functional concerns are classified as crosscutting if they are considered as a global system property.
- Linking goals to aspects: This method proposes that satisfaction of OR decomposed goals in the KAOS model usually will lead to tangled representation [48]. Instead, these methods should be implemented in an aspect-oriented manner [48]. In this research aspect identification starts from code level followed by a KAOS representation.
- Identifying Candidate Aspects with i* approach: this approach combines the i* approach and UML approaches for concern elicitation and identification. A use case model is derived from adapted i* approach and a set of crosscutting concerns are identified. If a use case is used by more than one use case or if it is used to extend more than one use case, this is a candidate for a crosscutting concern.

Benefits

Improving requirement specification

Identifying and modularizing crosscutting concerns from early phases of software development improves traceability among requirements [45]. It also helps to identify and resolve possible conflicts among requirements [46].

Contribution to other SE process steps

Early identification and modularization of crosscutting concerns:

1. Improves traceability between requirements and other software artifacts through software development, maintenance, and evolution [46,45].
2. Makes it easy to assess change impact while considering the influence of cross cutting concerns during development and maintenance activities [45].

Reusability

Reusability of system components can be improved in aspect orientation, as it is possible to handle each NFR by a specific component, avoiding confliction in the specification of system elements [47].

3.1.2.3 Quality Model

ISO/IEC 9126 is a famous international standard which presents a software quality model [49]. This quality model presents quality from two different perspectives – internal and external quality, and quality in use view. Several efforts has been made for NFRs specification based on ISO/IEC 9126 such as the Software QUality In the Development process (SQUID) approach, and ISO/IEC 14598-3 standard [2]. However, the focus on these efforts is on the control and evaluation of quality requirements during software development rather than their identification. Chirinos et al. present a quality model based on the quality views of ISO 9126-1 standard, namely REquirements CLassification Model (RECLAMO) to identify, classify and formalize NFRs [2]. NFRs in RECLAMO are identified based on ISO 9126-1 quality model, and formalized in measurable terms. The result of identification and classification is documented in a quality requirements list, and classified according to each quality view. As another instance, Doerr et al. propose a systematic, experience-based method based on ISO 9126 standard to elicit, document, and analyze a minimal and complete set of measurable and traceable NFRs [5]. Their proposed approach is also explained and compared with MOQARE use case driven approach in [34]. The experience-based method distinguishes between two stages: The first stage is prioritization of the QAs and tailoring quality models which typically have a tree structure. Tailoring means that quality models from other projects (or even companies) are reused and tailored to the needs of the software system to be developed. The result is several reference checklists and templates that are tailored to the current project context. In the second stage, these checklists and templates are used for the actual NFRs elicitation, documentation, and analysis. ISO/IEC 9126 has some problems [15]:

1. Ambiguity of the “Functionality” term: If “Functionality” is interpreted as “functional abilities” then functional requirements could be a subset of quality requirements.
2. The high level of details needed in specifying quality requirements
3. Mostly focus on the computer system rather than the whole system.

Boegh introduces ISO/IEC 25030 as a new standard for quality requirements in which software requirements address either the software product or the software development process [15]. Software product requirements include functional, quality and managerial requirements. The main characteristic of the standard is requirements specification by providing a set of quality measures with associated target values. A software quality attribute is considered as a measurable property – e.g. response time. Software quality (sub) characteristic is then defined as a category of software quality attributes. A quality requirement is associated to a quality characteristic and is defined by ISO/IEC 25010 quality model [74].

Quality models are used in the High-Level Objective-based Policy for Enterprises (HOPE) framework in order to systematically align high level business-oriented quality requirements with the system-level specification offered by the web service standards [50]. In HOPE, non-functional business policies applicable to the software (i.e. system’s qualities) are

identified by analyzing the rules and regulations [50]. For instance, one such policy could be "People working with customer information must be authorized". These policies are automatically refined into system-level quality properties in two steps [50]: In the first step, they are stated as "quality objectives" based on the domain-specific quality models of the framework and the application's business entity model. In the second step, quality models are again used to refine quality objectives into web service policies.

Benefits

Improving requirement specification

- In REQLAMO conflicts, inconsistencies, or ambiguities among quality requirements for each quality view and among the quality views are identified and resolved [2].
- As quality models in [50] align business-oriented quality objectives with system-level quality requirements, they improve the level of understanding of requirements among development team and stakeholders which in turn improves their collaboration and communication.

Contribution to other SE process steps

- REQLAMO facilitates the discovery of quality requirements which results in a more effective requirement analysis [2].
- Formalizing NFRs in ISO/IEC 25030 standard in terms of measures and target values helps designers to make the right decision in choosing necessary NFRs to be implemented [15]. Additionally after software implementation is finished, comparing the measured quality attributes with target values helps to verify software fulfillment in quality requirements [15].
- The experience-based NFR method is helpful for NFR elicitations by providing a clear guidance. The elicited NFR are measurable and traceable [5,34].

Ease of Use

In the experienced based NFR method a clearly defined systematic process exists for NFR elicitation and specification [34].

Reusability

The experienced based method captures past results and domain expertise, and also makes it possible to reuse artifacts such as quality models, checklists, and template [34,5].

3.1.2.3 Patterns and Languages

The goal of these approaches is to discover and use patterns which recurred in requirement specification regardless of the specific domain, project, or company. Patterns are more likely to be distinguished in some NFRs such as safety and performance [6]. However it is not straightforward for all NFRs such as robustness [6]. Shahrokni and Feldt introduce ROAST framework for specification and verification of robustness requirements at different abstraction levels through their proposed pattern [6]. Their pattern is similar to the Performance Refinement and Evolution Model (PREM) which is specific to performance requirements [6]. Oikinomopolous and Giritzails

present a template for security pattern and use it with their UML formalization [51]. Their template is constructed based on the design pattern [75] template but applies some changes to it. The changes are modification of some fields – such as Pattern Name and Classification and addition of some more fields – such as constraints to hold more security relevant information.

Some prior research are on providing fromal specification patterns in order to help formulate requirements correctly in formal notation, some of which focus on NFRs. For instance, Gruhn and Laue present timed propoerty specification pattern system for real time requirements that helps to verify the correctness of a system using model-checking tool [52]. As another example, Konrad and Cheng propose a real-time specification pattern system to specify timing based requirements in a formal specification language [53]. They also develop a structured English grammar to facilitate the use of specification patterns. Bitsch proposes safety patterns for formal specification of safety requirements using pre-specified generic safety requirements [54]. In order to help developers understand the meaning of patterns, each pattern is explained in natural language. Grunske mentions that most NFRs - such as safety, security, reliability, and availability have a probabilistic nature [14]. He presents a specification pattern repository for these kinds of NFRs, namely Probabilistic Property Specification Templates (ProProST) based on an underlying probabilistic temporal logic language. He also proposes a structured English grammar which relates the natural language specification to their probabilistic temporal logical formula.

Dal Cin presents an approach for NFRs specification using a grammar for an English like language [55]. This grammar focuses on representing dependability and performance requirements. The approach introduces the stochastic quantitative requirement language (SQIRL) for NFRs specification, in which sentences are independent of the modeling language [55].

Benefits

Improving requirement specification

- The structured language proposed in [55] makes NFRs specification non-ambiguous and easy to communicate among different stakeholders.
- Similarly, formulating probabilistic NFRs based on the proposed pattern system in [14] leads to an unambiguous relationship between the natural language and formal representation of the NFRs.
- Specification of robustness requirements in ROAST framework [6] improves the completeness while enhancing the level of requirements verifiability.

Contribution to other SE process steps

- Formal specification of NFRs based on the proposed patterns in [52] makes them a source for automatic model checking.
- The proposed safety patterns catalog in [54] shows the safety requirements which could exist in general; therefore it can be used as a checklist to discover relevant safety requirements in a software system.
- The patterns proposed in [51] facilitate the analysis and verification of security-relevant properties. The pattern system proposed in [14] allows the probabilistic NFRs to be

specified correctly in a formal way which in turn facilitates applying probabilistic verification techniques.

- The approach proposed in [55] supports refinement of the specification within the context of the system model in design phase and also translating the system model to a notation that can be interpreted by an analysis tool.

Ease of use

ROAST framework provides a clear guideline for elicitation and specification of robustness requirements on different levels of abstraction [6]. This framework helps to interpret, specify, and verify robustness requirements in a uniformed way [6].

Ease of use is addressed in [14,52-53,55] by providing structured English grammar or a pattern system which let non-expert practitioners specify NFRs correctly.

Reusability

Reusability is improved in this research focus through capturing and using specification patterns. Since these patterns can be used as a library and guideline for elicitation and documentation of NFRs, there is no need to develop requirements from scratch. Proposed patterns and structured language for NFR specification capture and transfer expert knowledge and could help in learning how to formalize NFR specifications.

3.1.3 Combination of approaches

Fatwanto and Boughton combine the NFR framework, uses case driven approach, and their proposed scenario-based specification technique [57] to analyze, specify and model NFRs, in the context of translative model-driven development. This method is limited to NFRs that can be manifested as operational/functional features. The NFR frameworks are used to model NFRs using SIG. Use cases are used to analyze NFRs especially in their functional (operationalizable) features. They provide a comprehensive view of system by integrating all these features. The integration of SIG into the use case driven approach is achieved by establishing associations between use case elements and SIG elements. Use cases are described in more detail using scenarios. Scenarios specify events – which could be triggered by actors or the system and also the corresponding actions that respond to one or more arriving events. They should avoid including any decision related to system design in order to increase the use cases reusability.

Chung and Supakkul combine the NFR framework and use case driven approach in order to specify NFRs [39]. Instead of using use case as the source for context in all kinds of NFRs, they propose to associate NFRs at four different use case model elements to give a more precise context. These four association points are system boundary, actor, use case, and communication association. The NFR framework is used to represent NFRs as softgoals to be satisfied. Also in order to illuminate any redundant specification for common NFRs, some NFR propagation rules are proposed where equal or more strict form NFRs are propagated to applicable use case elements. Supakkul and Chung combine the NFR framework, use case driven approach, and UML profile in another research in order to specify FRs and NFRs [58]. NFRs in this approach are represented as soft goals, and are associated with appropriate use case model elements [58]. These

two notations are formally integrated by the proposed UML profile [58]. In this profile a meta-model is defined to represent concepts in the NFR framework, and the extension points in the meta-model are identified where the concepts from the two notations are integrated. In another research, Chung and Supakkul offer a goal-oriented pattern framework which is an extension to the UML meta-model [59]. Object oriented UML is used for the capture of FRs, while goal-oriented NFR framework is used for capturing the NFRs. In this framework patterns are a collection of model refinement tools each defining the generation of a single target model element. These patterns can be combined to form larger grained patterns. This knowledge capture can be incremental, from small grained parts of the system to larger grained parts up to a point in which the whole system can be define via these patterns [59].

Cysneiros and Leite propose a systematic integration of NFRs into entity relationship (ER) data model, with the use of Language Extended Lexicon (LEL) as an anchor in order to improve NFRs specification and also conflict detection between FRs and NFRs [60]. Lexicon symbols are used to construct both the NFR graph and the ER model; so a common entry in the lexicon will be the link between the NFRs and the ER diagram. This strategy is extended in by Cysneiros et al., presenting a framework for integrating NFRs into either ER or Object Oriented (OO) conceptual models, again using an LEL [61]. In another research, Cysneiros and Leite propose an approach [62] to enhance the integration process presented before [60-61]. In this approach software development process constitutes of two independent cycles –functional and non-functional views with some convergence points between them. In functional view, FRs are recorded in LEL and expressed through UML diagrams such as use cases and class diagrams. NFRs are also recorded in LEL. However, due to the abstract nature of NFRs, they cannot be handled completely using LEL. In order to address this problem, they extend LEL to help NFRs elicitation and also to handle dependency links among them. Additionally, they use the NFR framework to improve reasoning about NFRs interdependencies. In the NFR framework, NFRs are considered as soft goals to be satisfied. Each soft goal is decomposed to sub goals using a graph structure until requirement engineer decides that the soft goal is satisfied. In this way, a graph is build for each NFR and the set of these graphs represent the non-functional aspects of the system. After that, as mentioned above, possible interdependencies are identified and solved using LEL. The key point of this strategy is that LEL is used as an anchor to build both the NFR graph and the UML diagram, which in turn will smooth their integration. The strategy provides a systematic integration of NFRs into use cases and scenarios as well as class, sequence and collaboration diagrams [63].

Leite et al. combine their proposed scenario and LEL to specify FRs and NFRs [64]. Scenario structure in this approach is comprised of title, goal, context, resources, actors, episodes, exceptions and constraint entities [64]. In this structure, “constraint” attribute is used for capturing NFRs applied to context, resources and episodes. The precise definition of constraint in this approach is: “A scope or quality requirement referring to a given entity. It is an attribute of Resources, basic Episodes, or sub-components of Context”. Scenarios in this approach are constructed based on the vocabulary of the Universe of Discourse (UofD) which should be

presented in an LEL. So, the software application is described by a set of scenarios while its vocabulary is described by LEL.

Castillo et al. integrate AOSD concepts, classic requirement engineering notations, and ISO/IEC 25030 standard [56] in their Requirement, Aspects and Software Quality (REASQ) conceptual model. All the concepts will be written in a way to support reuse, representing the domains of aspect-orientation, software quality and requirement engineering respectively. The REASQ model will integrate these concepts to act as an umbrella to define an aspect-oriented quality requirement engineering process. This framework tries to establish a mapping between ISO/IEC standards and AOSD concepts.

Kakollu and Chaudhary propose a Z specification to capture both FRs and NFRs and their relation in a formal way [65]. They mention that use case classification suggested by Jacobson [76] is useful to better capture the relationship between use cases. Jacobson classifies use cases as abstract, concrete, base, generalization, extension, inclusion and aspect. In their specification, a use case consists of an initiator, actors, invariants (I), states (S), events (E), state properties (P), starting and final states. Business policies and rules are specified using invariants. NFRs are documented using aspect use case which describes attributes of a service rather than the service itself. Relationship between use cases is defined by a UseCaseDependency relation which is specialized into include, extend and crosscut. Crosscut relation enables developers to document the crosscutting concerns.

Benefits

Improving requirement specification

- Standards provided in the REASQ model [56] and scenarios in [57,64] improve the understanding of requirements among different stakeholders which in turn improves their interaction.
- Integration of NFRs into data models proposed in [60-63] improves NFR specification and also conflict detection between FRs and NFRs.
- NFR propagation rules proposed in [39] eliminate the need for redundant NFRs specification where equal or more strict form NFRs are propagated to applicable use case elements.

Ease of Use

The proposed approach in [57] takes advantages of both NFR frameworks and scenarios being simple, intuitive, and easy to understand.

Contribution to other SE process steps

- The proposed approach in [57] benefits from using NFR framework which can be adopted easily into requirement elicitation and analysis methods. Additionally, using SIG to model NFRs in this approach will facilitate the tradeoff between NFR elements to choose the ones that are going to be implemented for the system.
- Integrating NFRs into ER diagrams in [60-61] helps to understand their impact on database conceptual model design. Additionally as the data model contains explicit

constraints of the design space, this approach results in more systematic design decisions [60-61].

- The use of proposed strategy in [62,77] can lead to a more complete conceptual models. Additionally this strategy helps to avoid errors which root in not properly dealing with NFRs. This will result in a faster time-to-the-market process [62,77].

Reusability

The proposed goal-oriented pattern framework in [59] allows the capture and reuse FRs and NFRs knowledge.

3.1.4 Discussion

- As shown in the Figure 2, the research focus is mostly on "combination of approaches" 11/51 (22%) and "goal-oriented approaches" 10/51 (20%), followed by "use case driven approaches" 9/51 (18%). These results show that
    - Researchers are mainly interested in investigating and integrating different NFRs specification approaches in order to reach a more applicable approach.
    - Goal-oriented approaches is the second most popular group that improve requirement specification in many ways, like achieving a high level of requirement completeness, avoiding irrelevant requirements, improving readability, etc.
    - Use cases are a popular requirements specification technique; therefore it seems reasonable that solutions are provided to adapt use cases for NFRs specification. Modifying use case diagrams or using control cases as complementary form of use cases are some examples of these solutions.
- Although a wide range of approaches exist for NFRs specification, little attention has been paid to the benefits and drawbacks of the approaches. There are very few publications focused on comparison of different approaches. Moreover, the applicable domain of these approaches is not mentioned in most of the cases. These can be a threat to the usability of these approaches when it comes to the industry, as the industry needs to be convinced that this approach is suitable and beneficial. Additionally, the industry needs to choose approaches which fit its domain, instead of trying out all the different approaches which is impractical.
- As depicted in Figure 3, in the last ten years there is a steady trend in providing solutions for NFR specification.

3.2 RQ2. What type of solutions is represented in the studies on NFRs specification?

3.2.1 Results

Figure 4 shows that the studies on the specification of NFRs are mostly of the type "Solution Proposal" 41/51 (80%). As depicted in Figure 5, the contribution type is mostly "method" 27/51(53%) followed by "framework" 12/51 (24%) considering the point that a publication might provide multiple contributions e.g. both a tool and method. As it can be seen in Figure 6 most of

the publications 43/51 (83%) are focused on general domains, and only 9/51 (17%) are for specific domains. Figure 7 shows a map of existing research focusing on NFRs specification, distributed over type of contribution and research type.

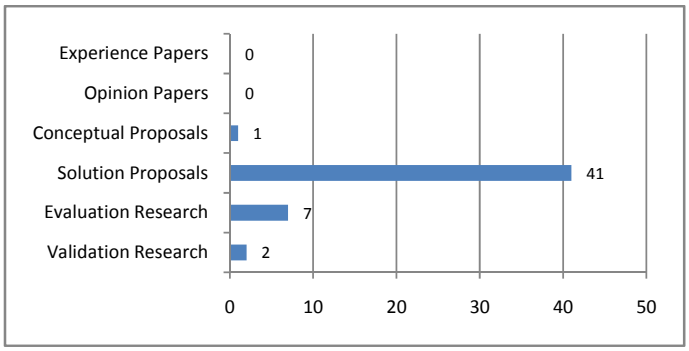

Figure 4. The distribution of research type –NFRs specification

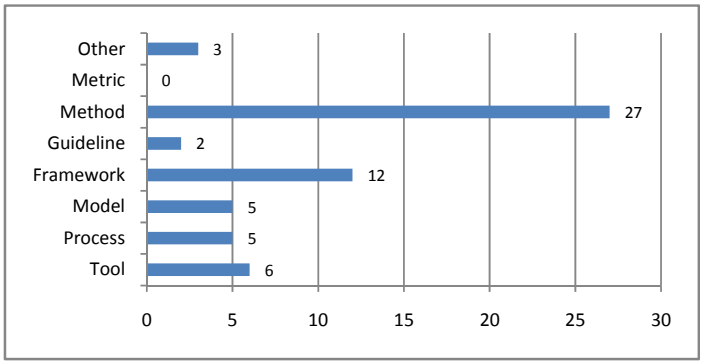

Figure 5. The distribution of contribution type –NFRs specification

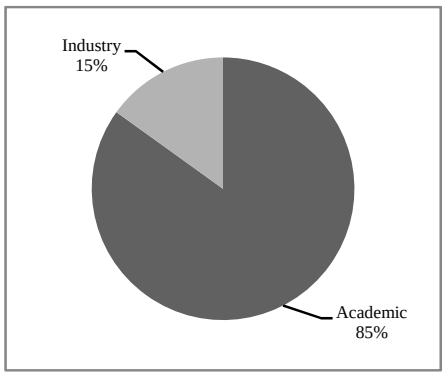

Figure 6. The distribution of publication context

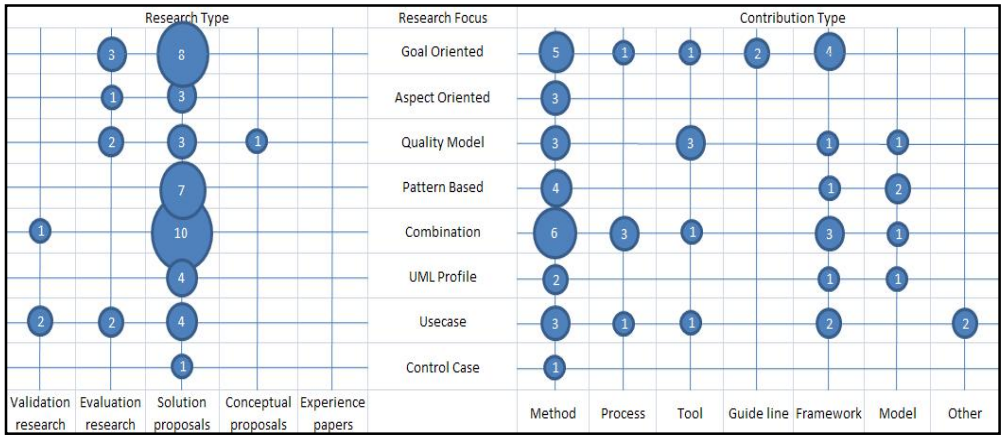

Figure 7. Map of research focus - NFRs specification

3.2.2 Discussion

- According to Figure 4, the main focus of publications is to propose solutions, while little effort has been concentrated on the evaluation of these solutions. This means that challenges in each research focus area are well understood, but the suggested solutions are just proposals with little focus on the actual use and evaluation of proposals. In the few cases where any evaluation has been done, investigating the solutions is mostly performed by the researcher him/her self and not by other independent researchers. While in reality a solution should be evaluated, and also tested in the industry before it can be used in an industrial scale. Hence there is high potential for future work on how to use and validate these proposals.
- As shown in Figure 5, the contribution type of publications is mostly “method”, followed by “framework”. This result encourages further research on presenting new methodologies, enhancing the existing ones, and providing supporting frameworks and tools in order to make the methods practical in industry.
- As a high percentage of the results are from academic research, given in Figure 6, there is a need for these ideas to be tested in the industry. There is also the need to carry out research on the methods which are used in the industry for NFRs specification.
- Figure 7 also reiterates the result mentioned previously that in all research focuses, the research type is mostly “Solution Proposal” and the contribution type is mainly “method”.

4. Conclusion

This paper presents a systematic review to investigate studies on NFR specification. A snowball sampling is also performed in combination with the systematic review which resulted in totally 51 relevant papers.

Below the major findings of the systematic review have been summarized:

- Combination of different approaches in order to benefit from the advantages of each participating method is shown to be popular in recent studies.
- Most of the publications that were uncovered during the review are “Solution Proposal”. Hence, challenges in NFRs specification area are well understood, however the proposed solutions are just proposals with little emphasis on the actual use and evaluation of proposals. In order for these solutions to be practical, complementary research for investigating and validating them is needed. Moreover, evaluating and implementing the solutions in industry should be taken into consideration to improve the quality of software systems.
- The contribution type of publications is mostly “method”, followed by “framework”. This result encourages further research into new methodologies, improvement of existing ones, and development of support frameworks and tools to make the methods applicable in industry.
- More research is required to compare existing approaches and also find applicable solutions for different domains.

5. References

1. Zave P Classification of research efforts in requirements engineering. In: Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on, 27-29 Mar 1995 1995. pp 214-216
2. Chirinos L, Losavio F, Matteo A (2004) Identifying quality-based requirements. Information Systems Management 21 (1):15-26
3. Zou J, Pavlovski CJ Control cases during the software development life-cycle. In: 2008 IEEE Congress on Services Part 1 (SERVICES-1), 6-11 July 2008, Piscataway, NJ, USA, 2008. 2008 IEEE Congress on Services Part 1 (SERVICES-1). IEEE, pp 337-344
4. Lindstrom DR (1993) Five Ways to Destroy a Development Project. IEEE Software 10 (5):55 - 58
5. Doerr J, Kerkow D, Koenig T, Olsson T, Suzuki T, Society IC Non-functional requirements in industry - Three case studies adopting an experience-based NFR method. In: 13th IEEE International Conference on Requirements Engineering, Proceedings, 2005. pp 373-382
6. Shahrokni A, Feldt R Towards a Framework for Specifying Software Robustness Requirements Based on Patterns. In: Requirements Engineering: Foundation for Software Quality. 16th International Working Conference, REFSQ 2010, 30 June-2 July 2010, Berlin, Germany, 2010. Requirements Engineering: Foundation for Software Quality. Proceedings 16th International Working Conference, REFSQ 2010. Springer, pp 79-84
7. Serrano A Capturing information system's requirement using business process simulation. In: 15th European Simulation Symposium and Exhibition, 2003.
8. Hussein M, Zulkernine M UMLintr: A UNIL profile for specifying intrusions. In: Riebisch M, Tabeling P, Zorn W (eds) 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Proceedings - MASTERING THE COMPLEXITY OF COMPUTER-BASED SYSTEMS, 2006. pp 279-286
9. Borg A, Yong A, Carlshamre P, Sandahl K The Bad Conscience of Requirements Engineering: An Investigation in Real-World Treatment of Non-Functional Requirements. In: Third Conference on Software Engineering Research and Practice in Sweden (SERPS'03), Lund, 2003. pp 1-8
10. Glinz M On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference. RE '07. , 15-19 Oct. 2007 2007. pp 21-26
11. Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007- 01
12. Cai KY, Card D (2006) An analysis of research topics in software engineering. Journal of Systems and Software 81
13. Chung L, Nixon B, Yu E, Mylopoulos J (1999) Non- Functional Requirements in Software Engineering. Kluwer Academic Publishers

14. Grunske L Specification patterns for probabilistic quality properties. In: Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on, 10-18 May 2008 2008. pp 31-40
15. Boegh J (2008) A new standard for quality requirements. IEEE Software 25 (Copyright 2008, The Institution of Engineering and Technology):57-63
16. Sindhgatta R, Thonse S (2005) Functional and non-functional requirements specification for enterprise applications. In: Bomarius F, KomiSirvio S (eds) Product Focused Software Process Improvement, Proceedings, vol 3547. Lecture Notes in Computer Science. pp 189-201
17. Onabajo A, Jahnke JH Modelling and reasoning for confidentiality requirements in software development. In: Riebisch M, Tabeling P, Zorn W (eds) 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Proceedings - MASTERING THE COMPLEXITY OF COMPUTER-BASED SYSTEMS, 2006. pp 460-467
18. Jaffe M, Leveson N Completeness, robustness, and safety in real-time software requirements specification. In: ICSE Proceedings of the 11th international conference on Software engineering, 1989.
19. Glinz M On Non-Functional Requirements. In: Requirements Engineering Conference, 2007. RE '07. 15th IEEE International, 15-19 Oct. 2007 2007. pp 21-26
20. Wieringa R, Maiden N, Mead N, Rolland C (2005) Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir Eng 11 (1):102-107. doi:10.1007/s00766-005-0021-6
21. Ivarsson M, Gorschek T (2009) Technology transfer decision support in requirements engineering research: a systematic review of REj. Requirements Engineering, vol 14, 2009, pp 155-175
22. Glass RL, Vessey I (1995) Contemporary Application Domain Taxonomies. IEEE Softw
23. Freelon D ReCal: reliability calculation for the masses. http://dfreelon.org/utils/recalfront/.
24. Goodman LA (1961) Snowball sampling. The Annals of Mathematical Statistics 32 (1):148-170
25. Chung L, Leite JCP (2009) On Non-Functional Requirements in Software Engineering. In: Alexander TB, Vinay KC, Paolo G, Eric SY (eds) Conceptual Modeling: Foundations and Applications. Springer-Verlag, pp 363-379. doi:10.1007/978-3-642-02463-4_19
26. Jürjens J Towards Development of Secure Systems Using UMLsec. In: FASE '01 Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering 2001. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp 187-200. doi:10.1007/3-540-45314-8_14
27. Jurjens J (2005) Secure System Development with UML.
28. Schneider G, Winters JP (2001) Applying Use Cases. In.
29. Adolph S, Bramble P, Cockburn A, Pols A (2002) Patterns for Effective Use Cases. In.
30. Laibinis L, Troubitsyna E Fault Tolerance in Use-Case Modeling. In: RHAS 2005 - the Workshop on Requirements for High Assurance Systems 2005.
31. Daniels J, Bahill T (2004) The hybrid process that combines traditional requirements and use cases. Systems Engineering 7 (Copyright 2005, IEE):303-319
32. Alexander I (2003) Misuse cases: use cases with hostile intent. Software, IEEE 20 (1):58-66
33. Sindre G, Opdahl AL Capturing Security Requirements Through Misuse Cases. In: Norsk Informatikkonferanse (NIK), Tromsø, Norway, 2001.
34. Herrmann A, Kerkow D, Doerr J Exploring the characteristics of NFR methods - a dialogue about two approaches. In: Requirements Engineering: Foundation for Software Quality. 13th International Working Conference, REFSQ 2007, 11-12 June 2007, Berlin, Germany, 2007. Requirements Engineering: Foundation for Software Quality. Proceedings 13th International Working Conference, REFSQ 2007. Springer-Verlag, pp 320-334
35. Herrmann A, Paech B (2008) MOQARE: misuse-oriented quality requirements engineering. Requirements Engineering 13 (1):73-86. doi:10.1007/s00766-007-0058-9
36. Lamsweerde AV (2001) Goal-Oriented Requirements Engineering: A Guided Tour. Paper presented at the Proceedings of the Fifth IEEE International Symposium on Requirements Engineering,
37. Plosch R, Mayr A, Korner C Collecting quality requirements using quality models and goals. In: 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC 2010), 29 Sept.-2 Oct. 2010, Los Alamitos, CA, USA, 2010. Proceedings of 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC 2010). IEEE Computer Society, pp 198-203

38. Lamsweerde Av, Letier E (2000) Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Trans Softw Eng 26 (10):978-1005. doi:10.1109/32.879820
39. Chung L, Supakkul S (2005) Representing NFRs and FRs: A goal-oriented and use case driven approach. In: Dosch W, Lee RY, Wu C (eds) Software Engineering Research, Management and Applications, vol 3647. Lecture Notes in Computer Science. pp 29-41
40. Jureta IJ, Faulkner S, Schobbens PY A more expressive Softgoal conceptualization for quality requirements analysis. In: Conceptual Modeling - ER 2006. 25th International Conference on Conceptual Modeling. Proceedings, 6-9 Nov. 2006, Berlin, Germany, 2006. Conceptual Modeling - ER 2006. 25th International Conference on Conceptual Modeling. Proceedings (Lecture Notes in Computer Science Vol. 4215). Springer-Verlag, pp 281-295
41. Castro J, Kolp M, Mylopoulos J (2002) Towards requirements-driven information systems engineering: the Tropos project. Inf Syst 27 (6):365-389. doi:10.1016/s0306-4379(02)00012-1
42. Fuxman A, Liu L, Mylopoulos J, Pistore M, Roveri M, Traverso P (2004) Specifying and analyzing early requirements in Tropos. Requir Eng 9 (2):132-150. doi:10.1007/s00766-004-0191-7
43. Giorgini P, Massacci F, Mylopoulos J, Zannone N (2004) Requirements Engineering Meets Trust Management: Model, Methodology, and Reasoning. In Proceedings 2nd International Conference on Trust Management (iTrust 2004)
44. Dardenne A, Lamsweerde Av, Fickas S (1993) Goal-directed requirements acquisition. Paper presented at the Selected Papers of the Sixth International Workshop on Software Specification and Design,
45. Rosenhainer L Identifying Crosscutting Concerns in Requirements Specifications. In: Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop at OOPSLA 2004, Vancouver,Canada, 2004.
46. Amirat A, Laskri M, Khammaci T (2008) Modularization of crosscutting concerns in requirements engineering. International Arab Journal of Information Technology 5 (2):120-125
47. Freitas EP, Wehrmeister MA, Pereira CE, Wagner FR, Silva ET, Carvalbo FC (2007) Using aspect-oriented concepts in the requirements analysis of distributed real-time embedded systems. In: Domer R, Gerstlauer A, Rammig FJ (eds) Embedded System Design: Topics, Techniques and Trends, vol 231. International Federation for Information Processing. pp 221-230
48. Bombonatti DLG, Melnikoff SSS Survey on early aspects approaches: non-functional crosscutting concerns integration in software systems. In: Recent Advances in Computer Engineering and Applications. 4th WSEAS International Conference on Computer Engineering and Applications (CEA 2010), 27-29 Jan. 2010, Athens, Greece, 2010. Recent Advances in Computer Engineering and Applications. Proceedings of the 4th WSEAS International Conference on Computer Engineering and Applications (CEA 2010). WSEAS Press, pp 137-142
49. ISO/IEC 9126-1:2001, Software Engineering—Product Quality—Part 1: Quality Model (2001).
50. Phan T, Han J, Schneider JG, Wilson K (2008) Quality-Driven Business Policy Specification and Refinement for Service-Oriented Systems. In: Bouguettaya A, Krueger I, Margaria T (eds) Service-Oriented Computing - Icsoc 2008, Proceedings, vol 5364. Lecture Notes in Computer Science. pp 5-21
51. S. O, S. G Integration of Security Engineering into the Rational Unified Process. In: Workshop on Specification and Automated Processing of Security Requirements, Austria, 2004.
52. Gruhn V, Laue R (2006) Patterns for Timed Property Specifications. Electron Notes Theor Comput Sci 153 (2):117-133. doi:10.1016/j.entcs.2005.10.035
53. Konrad S, Cheng BHC (2005) Real-time specification patterns. Paper presented at the Proceedings of the 27th international conference on Software engineering, St. Louis, MO, USA,
54. Bitsch F (2001) Safety Patterns - The Key to Formal Specification of Safety Requirements. Paper presented at the Proceedings of the 20th International Conference on Computer Safety, Reliability and Security,
55. Dal Cin M Structured language for specifications of quantitative requirements. In: Proceedings of HASE 2000. 5th IEEE International Symposium on High-Assurance Systems Engineering, 15-17 Nov. 2000, Los Alamitos, CA, USA, 2000. Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000). IEEE Comput. Soc, pp 221-227
56. Castillo I, Losavio F, Matteo A, Boegh J (2010) REquirements, aspects and software quality: The REASQ model. Journal of Object Technology 9 (4):69-91
57. Fatwanto A, Boughton C Analysis, specification and modeling of non-functional requirements for translative model-driven development. In: 2008 International Conference on Computational

Intelligence and Security, 13-17 Dec. 2008, Piscataway, NJ, USA, 2008. 2008 International Conference on Computational Intelligence and Security. IEEE, pp 405-410
58. Supakkul S, Chung L (2005) A UML Profile for Goal-Oriented and Use Case-Driven Representation of NFRs and FRs. Paper presented at the Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications,
59. Chung L, Supakkul S Capturing and Reusing Functional and Non-functional Requirements Knowledge: A Goal-Object Pattern Approach. In: Information Reuse and Integration, 2006 IEEE International Conference on, 16-18 Sept. 2006 2006. pp 539-544
60. Cysneiros LM, Sampaio do Prado Leite JC Integrating non-functional requirements into data modeling. In: Requirements Engineering, 1999. Proceedings. IEEE International Symposium on, 1999 1999. pp 162-171
61. Cysneiros LM, do Prado Leite JCS, de Melo Sabat Neto J (2001) A Framework for Integrating Non-Functional Requirements into Conceptual Models. Requirements Engineering 6 (2):97-115. doi:10.1007/s007660170008
62. Cysneiros LM, Leite JCSdP (2001) Using UML to reflect non-functional requirements. Paper presented at the Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada,
63. Cysneiros LM, do Prado Leite JCS (2004) Nonfunctional requirements: from elicitation to conceptual models. Software Engineering, IEEE Transactions on 30 (5):328-350
64. do Prado Leite JCS, Hadad GDS, Doorn JH, Kaplan GN (2000) A Scenario Construction Process. Requirements Engineering 5 (1):38-61. doi:10.1007/pl00010342
65. Kakollu DP, Chaudhary BD A Z-specification of classification and relationships between usecases. In: 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), 6-8 Aug. 2008, Piscataway, NJ, USA, 2008. 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD). IEEE, pp 779-784
66. Kulak D, Guiney E (2000) Use Cases: Requirements in Context. ACM Press
67. Cockburn A (2001) Writing effective use cases. Addison-Wesley,
68. Yu ESK (1997) Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Paper presented at the Proceedings of the 3rd IEEE International Symposium on Requirements Engineering,
69. Zou J, Pavlovski C (2008) Control case approach to record and model non-functional requirements. Information Systems and E-Business Management 6 (1):49-67. doi:10.1007/s10257-007-0057-x
70. Robinson WN, Elofson G (2004) Goal directed analysis with use cases. Journal of Object Technology 3 (5)
71. Filman R, Elrad T, Clarke S, Aksit M (2005) Aspect-Oriented Software Development. Addison Wesley, Boston
72. Chitchyan R, al. e Survey of Aspect-Oriented Analysis and Design Approaches. Technical Report, AOSD-Europe-ULANC-9
73. Jacobson I, Ng P (2004) Aspect-Oriented Software Development with Use Cases. New York: Addison Wesley Professional
74. ISO/IEC CD 25010, 2007. Software Engineering. Software Product Quality Requirements and Evaluation (SQuaRE). Quality Model and guide.
75. Gamma E, Helm R, Johnson R, Vlissides J (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley
76. Jacobson I (2004) Use cases – Yesterday, today, and tomorrow. Software and Systems Modeling 3 (3):210-220. doi:10.1007/s10270-004-0060-3
77. Cysneiros LM, do Prado Leite JCS Non-functional requirements: from elicitation to modelling languages. In: Proceedings of the 24th International Conference on Software Engineering,ACM, 2002 2002. pp 699-702

Testing of Non-functional requirements

A Systematic Literature Review

Zeinab Alizadeh Barmi

barmi@student.chalmers.se

Amir Hossein Ebrahimi

amirho@student.chalmers.se

Robert Feldt

robert.feldt@chalmers.se

Department of Computer Science and Engineering

Chalmers University

Goteborg, Sweden

Abstract

The increasing market demand for high quality software calls for software systems that meet their non-functional requirements (NFRs) in addition of implementing the desired functionality. Thus, to enable high product quality, it is crucial that NFRs are well specified and thoroughly tested in parallel with functional requirements (FRs). Nevertheless, in testing requirement the focus is mainly on FRs rather than NFRs. This study presents a systematic literature review on NFR testing in order to identify the current state of the art and needs for future research. The systematic review summarizes the 24 relevant papers found and discusses them within five major categories with "model based testing" and "component based testing" being the ones with most prior results.

1. Introduction

Software testing is a crucial part of software development that can directly determine software quality and project success. Grossmann et al. define software testing as: "any activity aiming at evaluating attributes or capabilities of programs or systems, and finally, determining if they meet the requirements" [1]. Software testing can be split into two categories, white box testing and black box testing [2]. White box testing focuses on the software structure and code implementation, whilst black box testing focuses on the software functionality, independent of the software implementation [2]. Efficient testing activities are necessary in development of software projects since not only do they lead software faults discovery, but also they are helpful to evaluate code reliability or performance of the software [2]**.** Testing is however a difficult and time consuming practice that becomes more challenging as systems grow in size and complexity [3]. Hence, testing constitutes a substantial cost, requiring man power, preferably automated tools and other resources [4]. However, if continuous testing is neglected during the development cycle,

fault correction cost would drastically increase with as much as 100 times compared to fault identification and correction before software release [2].

Requirements can be split into FRs and NFRs. FRs define the desired functionality of the system whilst NFRs define overarching attributes of the system, e.g. safety, performance, robustness, etc. There is a broad consensus in the software community that testing of FRs for a system without considering the NFRs is not sufficient to meet the needs of the customer. NFR testing is particularly crucial for safety critical software systems, e.g. medical diagnostics-, space- or air traffic control-systems, since these systems have more stringent demands. Testing of NFRs is however not restricted to safety-critical systems and the ever growing market demand for high quality software calls for efficient testing of less critical systems as well, e.g. web services. However, although NFR fulfillment is often more important than FR fulfillment for customer satisfaction [5], NFRs are generally verified late during the development cycle, e.g. after implementation or during integration [6]. Moreover, current testing methods and tools focus on the FRs and often even lack support to test NFRs [7]. This lack of support has resulted in many software projects failures. One example being the London Ambulance System where NFR non-compliance resulted in deactivation of the system immediately after its deployment [8]. Other examples are the "Moose-test" of the Mercedes Benz A Class and the Siemens mobile phone 65 series that experienced quality problems due to NFR non-compliance [9]. The reoccurring issue behind these problems is that NFRs are often tested separately from the FRs [10]. Another issue is that NFRs are often considered as technical issue in different development phases, before they have been adequately specified, defined or even properly understood [11].

Prior research on NFR testing can be divided into two categories. Studies in the first category focus on testing one or more specific NFRs, e.g. testing of performance requirement. The second category includes studies that are not limited to specific NFRs and rather provide solutions for testing of NFRs in general. Motivated by the search for studies belonging to the second group, we present a Systematic Literature Review (SLR) with the purpose to objectively identify, analyze, and compare the solutions proposed for NFR testing. The SLR is based on the guidelines proposed by Kitchenham [12] that define a SLR as:"a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest" [12]. Furthermore, we investigate which contribution types e.g. method, framework are given and, based on the classification by Wieringa [13], which research types e.g. solution proposal are used to test NFRs.

The rest of the paper is structured as follows: Section 2 describes the systematic review process. The results of the review are presented and discussed in relation to the research questions in section 3. Finally the conclusion is presented in section 4.

2. Research Methodology

In this section we explain the design and execution of the SLR. Figure 1 shows the steps of the systematic review that will be described in detail in the following subsections.

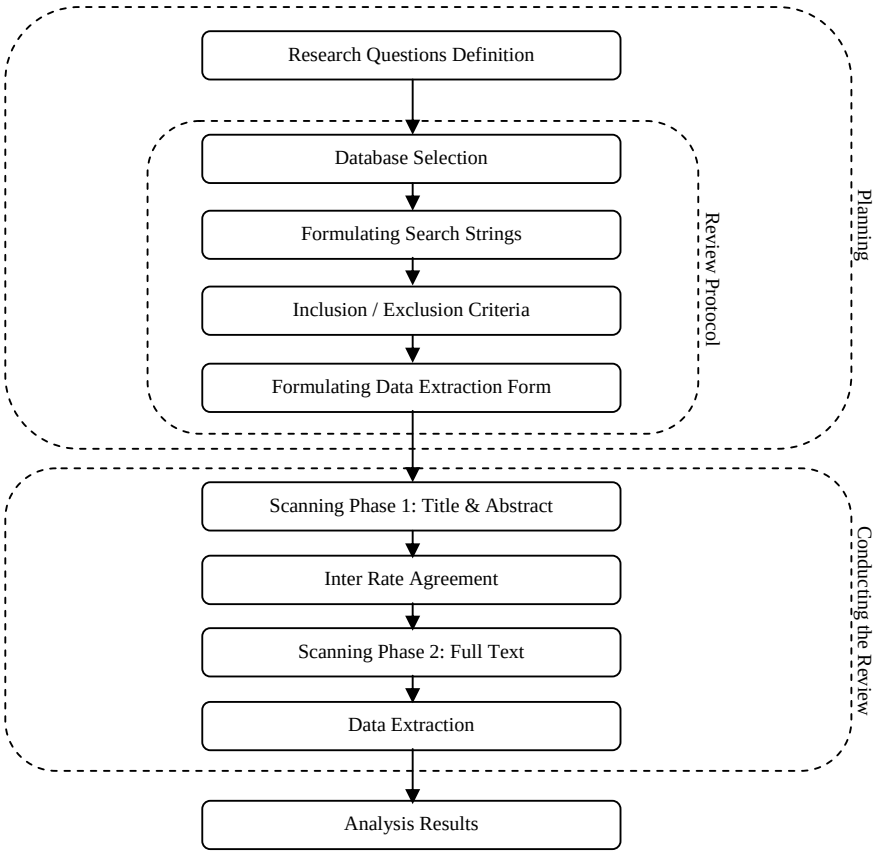

Figure 1 - Systematic Review Steps

2.1 Definition of research questions

The most important and pivotal part of any systematic review is to define precise research questions [14]. This SLR aims to investigate studies on NFR testing. In order to carry out this investigation the following research questions has been defined:

ID	Question	Aim
RQ1	Which studies have been done on testing of NFRs?	We want to find out which topics have been investigated and to what extent? What are the different perspectives that address testing of NFRs? e.g. model based testing, standards, languages, etc.
RQ2	What types of solutions is represented in the studies on NFR testing?	We want to find the contribution types given in different topics that address NFR testing (such as method, process, framework, tool, etc) and also research type (such as solution proposal, evaluation research, validation research, etc). We also aim to find out to what extent the evaluation of these solutions has been discussed.

Table 1. Research questions

2.2 Developing Review Protocol

The following section will describe the review protocol used during the SLR. The purpose of review protocol is to decrease the potential of researcher bias and to make it possible to replicate the review [12]. The main steps of a review protocol are:

1. Select the source databases
2. Formulate the search string
3. Define inclusion and exclusion criteria
4. Formulate a data extraction form

To ensure high protocol quality in this review it was evaluated and revised by another author with expertise in conducting systematic reviews.

2.2.1 Database selection

The first step of the review protocol is to find appropriate databases in which to search. As the search result is the basis for the final result, after inclusion and exclusion criteria have been applied, the choice of databases is important. The chosen databases were: *IEEE Xplore Digital Library, SciVerse Scopus, Inspec Engineering Village, and ISI Web of knowledge*. *IEEE Xplore Digital Library* covers almost all important conferences, workshops, and journal papers, which are published either by IEEE or ACM. *SciVerse Scopus* claims to be the largest database of peer-reviewed literature and quality web sources. *Inspec Engineering Village* and *ISI Web of knowledge* are also globally accepted source of choice for software researches. In order to handle the high number of found publications, we decided to use a reference management system. We chose EndNote X3[1] that is an industry standard software tool for storing, managing, and searching bibliographic references.

2.2.2 Search string formulation

In order to formulate a qualitative search string some publications on NFR testing were first identified and analyzed through an exploratory search approach [15-21,5]. This approach provided an initial set of papers that were used for forward/backward referencing, i.e. identifying more papers that referred to, or where referred by, the initial set.

The search string was then formulated according to the following steps:

1. Identification of primary search keywords from the research questions and initial set of papers
2. Identification of alternative and synonyms of the primary search keywords
3. Categorization of the search keywords into two categories. The first category focused on NFRs (named C1) and the second category on testing of NFRs (named C2). The keywords in the two categories were then joined with AND operator to construct the search string.
4. Refinement of the search string

[1] http://www.endnote.com/enabout.asp

Application of the initial search string on the chosen databases resulted in an incomprehensible number of hits. Many of the hits were also irrelevant since they were from other domains, i.e. not software engineering. Therefore, constraints had to be added to the search string, e.g. only papers written in English, only publications between 2000 and 2010, etc. It should also be noted that since the search was done in November of 2010 not all 2010 papers will be included in the result. The search string was refined iteratively with additional constraints in five steps. In each iteration the results were analyzed to see if they contained papers from the initial set of papers on NFR testing. In the case that papers from the initial set were not included or the result contained irrelevant papers the search string was improved further. The final search string categories, given in Table2, construct the final search string that is C1 AND C2.

NFR – C1
"nonfunctional requirement" OR "nonfunctional requirements" OR "non functional requirement" OR "non functional requirements" OR "nonfunctional software requirement" OR "nonfunctional software requirements" OR "non functional software requirement" OR "non functional software requirements" OR "nonfunctional property" OR "nonfunctional properties" OR "non functional property" OR "non functional properties" OR "quality attribute" OR "quality attributes" OR "quality requirement" OR "quality requirements" OR "quality attribute requirement" OR "quality attribute requirements"
Testing – C2
"test" OR "testing" OR "verify" OR "verifying" OR "verification" OR "validate" OR "validating" OR "validation"

Table 2 - Search string categories

Application of the final search string on the chosen databases resulted in 416 papers**.** It should be noted that C1 category does not include keywords for each particular NFR, e.g. performance, availability. Instead it contains keywords corresponding to NFR in general i.e. a broader view of the domain. This decision is motivated by the fact that there is no general consensus in the requirement engineering community on how to define, classify, or even represent NFRs [22]. Hence, it is not practical to cover all NFRs in the search string. However, if the search results in studies on testing specific NFRs, they will be accepted.

2.2.3 Inclusion-exclusion criteria

Inclusion/exclusion criteria are used to provide the research team members with a common understanding on the conditions under which a paper should be included or excluded from the SLR. The main criterion for inclusion in this SLR was that the paper should focus on NFR testing. Another inclusion criterion was that the paper had been peer-reviewed. Posters, Conference proceedings, short papers (with less than 6 pages), and opinion papers i.e. papers that express the personal opinion of author [13], were all excluded.

2.2.4 Formulating Data Extraction Form

The data extraction form developed for this SLR was based on the form developed by Ivarsson and Gorschek [23]. Some Keywords like context and the domain were formulated through analysis of the initial set of papers.

2.3 Selecting Studies

In the first step of the scanning process, all research team members scanned the title and abstract (if needed) of the search results based on the inclusion and exclusion criteria. Papers that were uncertain if they should be included or excluded were subject to a second scanning. This step of scanning process resulted in inclusion of 40 out of the 416 papers. During abstract scanning the original data extraction form was expanded with more key attributes. The key attributes of the data extraction form, their values and descriptions are shown in Table 3.

Key attributes	Values	Description
Research focus	service based testing, model based testing, component based testing, search based testing, and standard and languages	Defines different approaches to NFR testing from different focus groups.
Contribution type	tool, process, model, framework, guideline, method, metric and other	Specific contribution types and other details, e.g. the name of a tool, framework, etc.
Focused quality requirement or attribute	from the reviewed literature, general	This key attribute states if the research is focused on one specific quality requirement, otherwise marked as general.
Research type	validation research, evaluation research, solution proposals, conceptual proposals, opinion papers, and experience papers	The classification of research type is based on the classification proposed in [13].
Context	academia, industry, and open-source software	Whether the research is performed in academia or based on results or experiences from industry [23].
Domain	real-time systems, safety-critical systems, web systems, information systems, process controlled systems, banking and finance, education, energy resources, government and military, insurance, medical/health care, telecommunication services, transportation, other	This key attribute states if the research is done for a specific application or domain, otherwise marked as general. The values for this property are taken from Digital's Industry Taxonomy [24].
Scale	toy example, down-scaled real example, industrial, N/A	This key attribute defines the scale of the study. The values range from toy examples used in student projects to

		industrial scale applications [23].
Benefits and drawbacks	from the reviewed literature, N/A	This key attribute defines the benefits and drawbacks of the proposed approach that are mentioned in the research.
Perceived industrial benefits	from the reviewed literature, N/A	This key attribute defines the industrial benefits of the proposed approach that are mentioned in the research.
Context described	weak, medium, strong	If no context is provided in the study the value of this key attribute would be weak. The medium value is given if the organization/company and development efforts for the study are briefly described. If the organization/ company and development efforts are described in detail the strong value is given to the attribute [23].
Evaluation discussed	weak, medium, strong	This key attribute shows the degree to which the evaluation of the study is discussed. The evaluation value is set to weak if the paper does not discuss the evaluation of the study. If the evaluation is mentioned but not discussed in detail the value is set to medium. Otherwise, if the paper discusses the evaluation in full the value is set to strong [23].

Table 3 - Key attributes of data extraction form

Before proceeding to the second step of the scanning process an inter rate agreement was defined to ensure that all research team members had the same understanding of the inclusion/exclusion criteria. The inter rate agreement was calculated based on each team member result for randomly selected subset of all the papers, 10 %, using ReCal online utility [25]. The calculated inter rate agreement (92.9%) was at an acceptable level.

In the second step of the scanning process the full text of the papers was analyzed during which the data extraction forms were filled for each paper. If the team members did not agree on the inclusion or exclusion of a paper it was discussed until a decision was made by consensus. At the end of this step 24 out of the 40 papers from the first scanning step remained.

2.4 Validity Threats

The main threats to validity of the SLR are publication and selection bias, and data extraction, each detailed below:

2.4.1 Publication and selection bias

There are some threats to validity in the process of paper selection:

- Inclusion of papers based on the information provided in the title/abstract could introduce a threat as title/abstract of a paper might not reflect its actual content. This threat was found to be limited as for papers in which testing of NFRs is not mentioned in the title/abstract, it is more likely that this issue is not the main contribution of these papers.
- The decision for inclusion/exclusion of papers is subjective. Although paper selection process is carried out on the basis of criteria to minimize such a threat, the use of these criteria could still prove subjective. Three steps were taken to limit this potential threat. First, a pilot paper selection was performed (on 100 papers) before the actual process, based on inclusion/exclusion criteria, in which research team members aimed to reach a common understanding of the inclusion criteria. Second, title/abstract scanning was performed by all team members in parallel and the outcome was cross-checked. In case of disagreement those specific papers were discussed, if a mutual result was reached the disagreement would be resolved otherwise this paper was left for the detailed study in the next step. Third, after the first step of the scanning process an inter rate agreement was calculated to ensure that all team members had an acceptable level of understanding of the selection criteria.
- Relevant papers that are published before our publication year constraint will be missed. After investigation, this threat was also found to be limited as the majority of the research before this period were focused on FRs.

2.4.2 Data extraction

Assignment of values to different key attributes of the data extraction form could be subjective. In order to mitigate this threat each research team members analyzed the selected papers of the pilot paper selection and filled the data extraction forms. Then the resulted data extraction forms were compared to find the necessary modifications of values/descriptions of key attributes, and also to reach a common understanding among team members.

3. Results and Analysis

The following sections outline the results and analysis for each research question.

3.1 RQ1. Which studies have been done on testing of NFRs?

In order to identify the different perspectives with which testing of NFRs are addressed, the results of the systematic review were classified according to their research focus, publication year, and other attributes. These attributes aimed to help answer RQ1. In Table 3, the publications are classified by their research focus. The distribution of research focus is given in Figure 2. Figure 3 shows the distribution of research focus over different years. Benefits and drawbacks of studies in each research focus are listed at the end of each corresponding section. It should be noted that the benefits and drawbacks are specifically stated in these papers and are not the result of our interpretation.

Research Focus	Papers
Service based testing	[26],[27], [28]
MBT	[2],[3], [29], [30], [31], [32], [33], [34], [35], [36]
Component based testing	[4], [7], [10], [37], [38], [39]
Standard and Languages	[1], [2], [40], [29], [41]
Search based testing	[42], [43]

Table 4 - Publications classified by the research focus - NFR testing

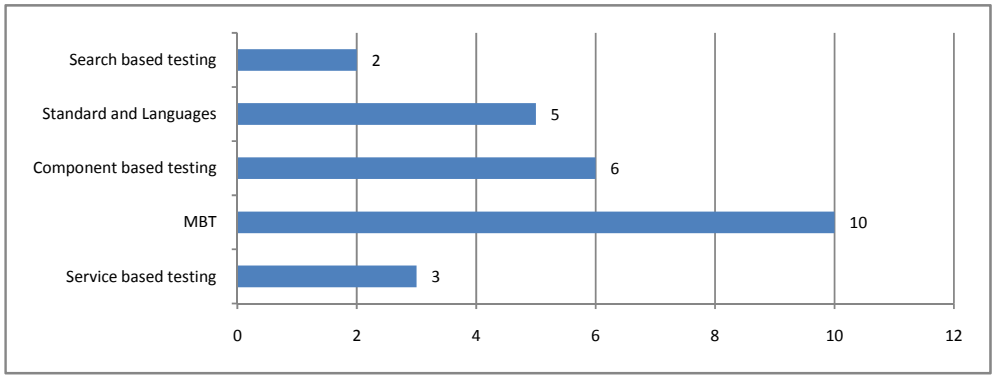

Figure 2. The research focus distribution of the publications - NFR testing

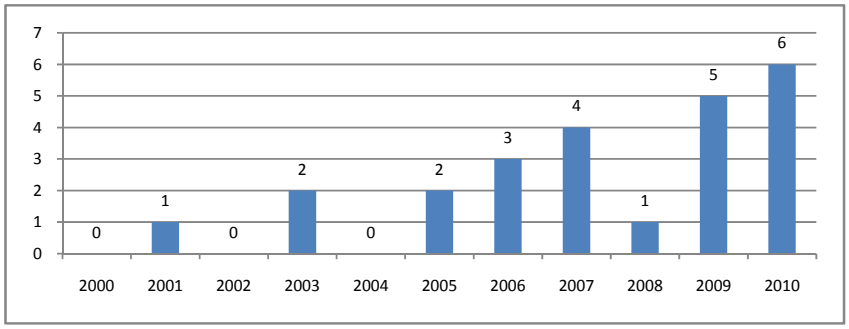

Figure 3. The distribution of publications according to year - NFR testing

3.1.1 Model based testing (MBT)

The idea behind Model Driven Engineering (MBE), introduces by the OMG's initiative, is to use models as the primary engineering artifacts to effectively represent and resolve any domain specific problem [31]. These techniques aim to handle both static and dynamic aspects of a software system during its development life cycle starting from requirement specification to software implementation and testing [29]. To address the challenges of software testing in MBE, MBT techniques has been proposed to use models for testing and code generation [33]. While the

focus of these techniques has been primarily on testing of FRs, recent approaches also encompass testing of NFRs.

MBT includes four main phases [3]: build the model, generate test cases, execute test cases, and check the conformance of test results to the system requirements. Test cases are generated based on either a behavioral specification or usage models derived from the system's requirement specification [29]. Usage models show all user scenarios of a software system at a given level of abstraction for FR and NFR testing. In [2,29] a Markov Chain Usage Model (MCUM) is generated based on the requirements specification and is used to derive Testing and Test Control Notation (TTCN-3) test case definitions. Another example is Lencevicius et al. that propose the validation of a mobile devices' power consumption, e.g. minimum power level and energy consumption, through modeling device power [30]. Devices, as power consuming components, are modeled in a hierarchical state transition diagrams in which each state and transition has its own power level function. When a global system event occurs, all concurrent parts of the state transition diagram are triggered and cause transitions between the power states. These transitions can be observed and thereby used for requirement validation. Additionally, this approach provides test cases for validation of the completeness and correctness of event traces using extended message-sequence charts.

Some MBT techniques establish or modify languages for NFRs specification and testing, e.g. developing a Domain Specific Language (DSL) for the specification and compliance verification of Service Level Agreement (SLAs). SLAs are used to specify quality requirements, e.g. performance, for services [31]. However, SLAs are specified using natural language and are therefore not suited for automatic verification. Correia et al. solved this problem by developing a DSL to specify quality requirements in SLAs at the problem domain abstraction level. The meta-model for the DSL is based on the meta-model of process modeling language of IT services (BPMN). The resulting meta-model is verified through an Object Constraint Language (OCL) interpreter that can run statements written in the SLA language. Another example of language modification is given by Jasko et al. [35], where test viewpoints are added to the User Requirements Notation (URN) that is an International Telecommunication Union (ITU) standard [35]. In URN, FRs are modeled in Use Case Maps (UCM) and NFRs with the Goal-oriented Requirement Language (GRL). In order to add test viewpoints to the URN, business or system goals and their implementation strategy are defined in GRL by managers in the software planning phase. By combining GRL with UCM test preparation can be conducted, e.g. generation of test purposes [35].

Quality attributes of distributed software systems such as performance, security, and correctness can also be improved with MBT, as summarized by Saifan and Dingel in their survey [3]

1. Performance: in order to validate that a distributed system meets the performance requirements, testers should consider all performance characteristics of the system during performance testing, e.g. latency, throughput, and scalability. Various approaches have been proposed to derive performance models from software architectures, primarily through UML diagrams. Some of these performance models are: Layered Queuing

Network (LNQs), which are an extension to the Queuing Model (QM) [44], Stochastic Petri Nets (SPNs) [45], or Stochastic Process Algebra (SPA) [46].

2. Security: models for security requirements have also been subject to research. For instance, UMLsec [47-48] that is an extension of UML diagrams uses stereotypes with tags and constraints to insert security requirements into the UML diagrams. This approach allows detection of violations to the security requirements in UML diagrams.
3. Correctness: models can also be used to check the correctness of a system under test by comparing the actual system outputs with the expected outputs using conformance relations, e.g. input-output conformance. In general, conformance relations are based on comparing traces, i.e. sequences of states that the system model exhibit during execution. However, it is not sufficient to simply compare the initial and final execution states in a distributed system since these systems may be designed never to terminate. Another limitation with this approach is that correctness may depend on the order in which messages are exchanged that is complex to model.

Hussain and Eschbach focus on model-based testing of safety requirements in software intensive networked automation systems [32]. Their approach is based on the automated generation of fault trees. The first step in generating a fault tree is to build and compose formal models for components of the control software, components of the process that should be controlled, as well as the hardware and network failures. In the second step, model checking is performed on the formal models to check whether failure scenarios can cause hazards. Hazards are identified through failure injection in the model, normally by setting integer or Boolean parameters in the model. The result of the model checking is the fault tree and/or test cases that can be used for risk-based testing of safety requirements.

There are some tools that support model based testing of NFRs. Gregoriades and Sutcliffe have proposed a tool that is a considerable advance compared to other current MBT tools that are based on model inspection [34]. The tool explicitly considers and visualizes the environmental influences on software system, based on its requirements, in order to pinpoint problematic tasks and components in the scenario and design sequence [34]. The tool is applicable to problems where NFRs are expressed as properties of components such as the human and machine agents in system engineering domains [34]. SpecExplorer [49] and TorX [50] are other examples of MBT tools for testing of distributed systems [3]. SpecExplorer is the second-generation of MBT technology, developed by Microsoft, and is used to test, among others, operating system components and reactive systems. TorX is a tool that implements the input-output conformance theory. In this tool there are four main phases that are: test case generation from system specification, translation of test cases into executable test scripts, test case execution, and test analysis.

Using a single model to specify and test both FRs and NFRs is perceived as beneficial, according to Corriveau [33]. In his research on MBT, in offshore outsourcing systems, Corriveau states that using more than one model will increase the risk of misinterpretations as well as:

1. Communication risks, i.e. the more models, the higher the chance of miscommunication. Especially since more syntax and semantics have to be learnt by the two communicating parties.
2. Conformance testing risks, i.e. the more models, the more complicated the testing task will become [33].

Corriveau proposes a set of attributes for the semantic vehicle used to capture and test quality requirements. These attributes are strong testability, executable tests, responsibilities and scenarios, coverage, and modeling a continuum of levels of abstraction [33].

Misuse cases are a negative form of use cases that can be considered an MBT approach in which the possible misuses of a software system are predicted and modeled [36]. A misuse case is defined as ‘‘a function that the system should not allow’’ [51]. Misuse cases have been shown to be efficient for specification and testing of NFRs whose goal is to handle software threats. One such NFR is security requirements, since they specify prevention or mitigation of threats from negative agents that seek to misuse or cause the system to fail [36]. Other NFRs, such as safety, reliability, usability, maintainability, portability, and hardware aspects like storability and transportability, can also be specified and tested by misuse cases [36]. This approach can be used as long as negative agents and threats related to these agents can be identified. For instance, negative agents for reliability requirements are human errors or design errors that can cause the system to fail. Another example of a negative agent, for usability requirements, is a confusing user interface [36].

Benefits

- Models have generally been used to facilitate communication between developers and stakeholders and also among different stakeholders [33]. One of the advantages of using models is the level of abstraction that they provide [33]. For instance, using a DSL allows the solution to be expressed on the same level of abstraction as the problem domain that in turn helps domain experts to easily understand, modify and validate the specifications at the domain level [31]. Additionally, requirements compliance validation on the same level of abstraction as the service specification, e.g. [35], would make it easier to understand by stakeholders.
- The URN approach has the following advantages [52]
    - Test preparation does not require an exact interface definition.
    - Early test planning helps to save time and cost during software development.
    - The approach increases the level of collaboration between developers and test engineers.
    - It is possible to link an UCM to a performance test language, like TTCN or LNQ
- The proposed approach in [32] for testing of safety requirements in software intensive networked automation systems helps to improve the quality of a system with the help of fault trees. Fault trees provide improved traceability between testing activity and safety requirements, but also completeness and correctness of requirements.

- Misuse cases proposed in [36] for MBT, are simple, easy to understand and apply. They can also be used efficiently for requirements elicitation, analysis, design trade-offs, and verification [36].

3.1.2 Standards and languages

To help companies to improve their testing of NFRs both languages and standards have been developed. TTCN is one such language that is currently on version 3, i.e. TTCN-3, that has been standardized by the European Telecommunication Standardization Institute (ETSI) [29]. TTCN-3 is a test specification language that was originally developed for testing telecommunication systems [1]. However, it can be applied in any application domain, e.g. for testing web service or module testing [29]. Dulz has proposed that a TTCN-3 test suite can be derived from an UML 2.0 requirements model through the process described in the TestUS framework [29]. This process starts with refining UML sequence diagrams by adding special annotation tags that conform to the UML SPT Profile. The tags include schedulability, performance and time. The result is a Quality of Service (QoS) requirements specification. In the next step of the process, a MCUM is automatically constructed from the annotated sequence diagrams from which an executable test suite can be constructed on-the-fly, i.e. in runtime. The purpose behind the technique is to avoid the long compilation times associated with development of TTCN-3 test suites for real-world systems.

Currently TTCN-3 suites are used to test real-time systems and research is being conducted to improve the technique for that purpose. For instance, Grossmann et al. propose an extension of TTCN-3 in order to support testing of time related NFRs in real time systems. They state that TTCN-3 is not applicable for real time systems since it does not provide any control of what happens between two instructions in a test [1]. Therefore, they propose that TTCN-3 should be extended with two new abstract data types which are datetime and timespan. Datetime specifies event timing and timespan specifies the time delay between events. The extended version of TTCN-3 is used in combination with the Automotive Open Systems Architecture (AUTOSAR) platform for functional testing of basic components. TTCN-3 is also used to test FRs and NFRs of time-critical systems in the European IST project Markov Test Logic (MaTeLo) [2]. In the MaTeLo project a MCUM is automatically generated from a Formal Description Technique (FDT) description from which TTCN-3 test case definitions are derived. In order to specify non functional properties, which are not supported by FDT, MaTeLo uses the Corsair extension approach [2]. In this project, testing consists of a combination of statistical-usage testing and specification-based testing. In statistical-usage testing, test cases are specified based on the expected operational usage of the system, from users external point of view, to assure independence from the implementation [2]. Statistical-usage testing allows the MCUM to be used to automatically derive both non-random test cases e.g. model coverage tests, but also statistical random test cases. Specification-based testing allows the MCUM that is constructed based on the annotated sequence diagrams to be used to automatically derive test cases [2]. These test cases can also be used to test non-functional properties by extending three types of TTCN-3 test verdicts, PASS, FAILED and INCONCLUSIVE [2].

Due to the many perceived advantages of TTCN-3 it has gained industrial appeal. For instance, Dei et al. adopted a three-phase study approach to deploy TTCN-3 at NOKIA [40]. In the first phase, basic tools were designed and used in pilot projects in a core team at the company. In the second phase, expert users were also involved to develop a test system for a product that had positive results. The positive results led to a broader application of TTCN-3 in several Nokia's business units and also at subcontractors of the company. The study result showed that this approach provides more efficient and productive test-case development than its predecessors [40].

ISO/IEC 12119: 1994(E) is a standard that includes a set of quality requirements for software packages and instructions on how they should be tested [41]. Test practices defined by this standard focus on document inspection and black box testing of quality requirements in software packages [41].

Benefits

TTCN-3 language is modular, well structured, standardized, easy to learn, flexible to use, and extendible with definitions from other languages [53-54]. NOKIA's end users report that TTCN-3's well-integrated tool chain allows the company to develop more complex test cases than they could with the previously used test languages [40]. Additionally, TTCN-3 as a unified testing technology, relieves their engineers from having to learn a new testing language when they have to move between test organizations [40]. Other advantages of TTCN-3 are:

- It supports several different types of testing, e.g. conformance, robustness, regression, system, and acceptance testing [29].
- It supports automated testing and provides abstraction from implementation details [40]
- It can be used for executable test case generation for many application domains due to its modern component-oriented inter-communication properties [2].

3.1.3 Search based testing (SBST)

Search Based Software Engineering (SBSE) has been shown to be a successful technique for software development starting from requirement engineering to software testing and maintenance [42]. SBSE also includes specific methods for testing, referred to as Search Based Software Testing (SBST) and Search Based Test Data Generation (SBTDG). The idea behind SBTDG is to model a test requirement as an optimization problem for a numeric function that can be solved with heuristics [42]. SBTDG can be used to test both FRs and NFRs and has been shown to be a solution to many of the problems related to current random and static testing approaches. These problems include test data generation failure for complex software and failure to comply with complex criteria in random approaches. SBTDG also supports test data generation for code constructs such as loops, pointers and arrays that are lacking in many current static approaches [42]. SBTDG is based on coverage criteria, e.g. function coverage, statement coverage, branch coverage, etc. However, Ghani and Clark argue that using branch coverage as the main coverage criterion is not strong enough for SBTDG. They therefore propose a SBTDG framework in which multiple condition and Modified Condition/Decision Coverage (MCDC) coverage are instead used to provide a stronger criteria [42].

The concept behind SBST is that software test generation is a non-deterministic problem due to the many variations of input to a system [43]. SBST solves this problem by using a test adequacy criterion that is described with a fitness function. The fitness function is then combined with a metaheuristic search technique to evaluate a set of possible solutions in the search space of possible solutions to find the best solution. The systematic review on search-based non-functional testing techniques conducted by Afzal et al. shows that SBST has successfully been applied to test execution time, quality of service, security, usability and safety [43]. SBST meta heuristics that were discussed in the review are: Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GAs), Ant Colony methods (ACO), grammatical evolution, genetic programming, and swarm intelligence [43]. The following section summarizes the results from the review.

Execution time testing: Evolutionary testing (using GAs) is a promising approach for testing of execution time, where the best and worst case execution times (BCET, WCET) are identified to determine whether a timing constraint has been fulfilled. GA is the most commonly used technique for this type of testing but also the SA technique has been used, found in one study during the review. The reason GAs have higher representation is because they provide a population of possible solutions that increase the possibility of locating an optimum solution. SAs on the other hand only provide one solution at a time that may or may not be the optimum solution.

Quality of service testing: SBST techniques have also been applied to address the QoS-aware composition problem and to find SLA violations between the service integrator and end users. The fitness function is in these cases used to maximize specific QoS attributes, e.g. reliability and availability.

Security testing: the goal of security testing is to detect security vulnerabilities, e.g. buffer overflows. SBST techniques that have been shown successful for this type of testing are grammatical evolution, linear genetic programming, GA, and particle swarm optimization.

Usability testing: usability requirements consider user interactions with the software system through its user interface. Usability requirement conformance is however difficult to test due to the large amount of possible interactions and related features of a software system. Many techniques, e.g. exhaustive testing, are therefore considered infeasible for the task. SBST solves this problem by using a coverage array (CA) and only test every combination of pair-wise (t-way) interactions at least once. The efficiency of this solution depends on producing smaller CAs in a reasonable amount of time. SBST is efficient for small parameter sets, however, for large parameter sets the information storage space grows too large and the technique becomes infeasible.

Safety testing: choosing the appropriate method to test safety requirements is crucial for safety critical systems. SBST techniques, SA and GA, are suitable for safety requirement testing since they can generate test data that violates safety requirements of the system under test and thereby expose potential weaknesses.

Limitations

- Evolutionary testing has some limitations for execution time testing:
    - Insertion of hardware dependent counting instructions affects the execution time [43]

    - Identification of the optimal solution from the generated execution times is difficult [43]
    - It is difficult to produce qualitative input parameters according to BCET; WCET for complex test objects. The ideal solution is therefore to combine the evolutionary testing with domain knowledge of the test object. Infusion of domain knowledge allows identification of search areas that are difficult to reach using information provided by the fitness function, thereby raising test coverage.
- QoS testing of service-oriented systems includes several difficulties, for instance the code or even architecture of a service are often not available that makes testing costly. SBST can resolve this lack of control but requires an appropriate metaheuristic, chosen based on the context. However, the metaheuristic is seldom a perfect fit for the problem and therefore has to be tailored, which is both complex and costly.

3.1.4 Testing NFRs in service based software engineering

Service based software engineering has changed the way one looks at the internet, from a repository of data to a repository of services, i.e. web services [26]. Web services allow companies to develop applications that can communicate with business partners and customer applications cross development languages and platforms [26]. The spread of web services is however limited due to lack of trust in the services abilities. This trust issue originates in the fact that customers only have access to the Web Service Description Language (WSDL) documentation but not the source code implementation [26]. The level of trust can be increased by improving the web services non-functional attributes, e.g. performance, reliability, etc. Additional NFR testing is therefore required to increase the confidence and trustworthiness of web services [28]. From a business perspective, NFR compliance can be a selling point to customers that have many services with similar functionality to choose from [28].

Yeom et al. state that QoS is a key factor in differentiating web services [27]. They define a multi-aspect QoS model to categorize web service qualities, including non-functional properties [27]. The model has three views, the business-level view, the service-level view (performance, stability), and the system-level view (interoperability, manageability, business processing, and security) [27]. Yeom et al. also define a framework for testing manageability of web services that includes a testing architecture, testing items and testing scenarios. The testing architecture is made up of a driver, a test assertion validator, a WSDL parser, manageable web services and a report [27]. In order to measure the manageability of a web service it should include a set of interfaces specified in WSDL [27].

A critical challenge of NFR testing for web services is that test results can be affected by the middleware or the web service deployment platform. In order to show this problem, Hanna and AbuAli deployed the same web service in the Axis, GLUE and .NET platforms and compared their robustness and security [28]. For security requirements, Axis proved to be the best platform with no security failures while .NET was the worst with 15 security failures. Their results also show that .NET was the best platform in terms of robustness, whilst GLUE was the worst [28]. In another study, Hanna and Munro state that web service platform or the middleware can affect

robustness of a web service. The reason is that web service platform on the provider side may intercept a Simple Object Access Protocol (SOAP) request before it reaches the web-service implementation if the request contains invalid data. Hanna and Munro therefore state that a distinction has to be made between robustness of the middleware and robustness of the web service implementation [26]. Their suggested approach is to develop robustness test cases based on analysis of the WSDL documents with fault based testing techniques to ensure web service quality [26].

Limitation

The body of knowledge on NFR testing of web services includes a few problems, as uncovered by Hanna and Munro [26]. These problems include:

- Some studies do not specify what web service quality attributes they are assessing
- Different naming conventions are used for web service quality attributes in different studies
- Some studies aim to increase the trustworthiness of web services but do not mention what quality attribute that aims to be improved.
- Some studies state that negative testing should be used to verify web service quality but do not mention how to generate negative test data to that end.

Hence, the applied testing techniques are not described on a satisfactory level.

3.1.5 Testing NFRs in Component Based Software Development (CBSD)

In Component Based Software Development(CBSD), components are defined as self-describing, reusable and modifiable units of functionality that can be combined to build a software system [37]. Modern enterprise software systems can be considered component based systems in which a system does not exist in isolation but is rather a component within a broad network of interacting systems [26]. These systems of systems vary greatly in size, are usually very complex, operate across different domains and can be distributed over large geographical distances [55]. Components can be validated at three key stages: during design through static analysis, when testing components during implementation, and when validating components in actual deployment scenarios [37]. Automated validation of components presents some unique challenges that require specific practices. The first challenge is to encode component constraint information in each component. Another challenge is component development that supports access of constraint information at run-time. Usage of constraint information to run exclusive and realistic tests on deployed components is considered as another challenge.

Aspect oriented techniques can be efficiently used to test FRs and NFRs of component based systems. The reason is because these techniques promote separation through modularizing of crosscutting concerns which would otherwise be scattered and tangled, making them difficult to test [7]. Additionally, aspect-orientation aids to facilitate automated testing through its flexible and modularized design [7]. The easy weaving of test components into the code architecture and the non-intrusive nature of test aspects makes this technique an attractive solution for non-functional requirements testing [7]. In one example study, testing is based on a codification mechanism that is

used to capture the impact of cross-cutting concerns on software components, thus describing the functional and non-functional constraints of the software [37]. Component aspects, e.g. constraints, are encoded in XML files, associated with component implementations. “Validation agents” can then query the XML encoded component descriptions to formulate and run tests [37]. Use of aspect orientation for testing performance , reliability, and robustness is proposed by Metsa et al. [7]. They state that this technique can be used to inject performance testing into already existing systems to observe memory consumption or to supervise critical execution times [7]. Properly formulated aspects provide a sophisticated method for reliability testing, which often involves repetition, data collection, statistics, and estimation [7]. In order to test system robustness, aspects can be used to generate test cases to verify system behavior in starvation situations, e.g. initial service request to be starved [7].

In their research, Metsa et al. report that aspects have been rarely used in unit tests [7]. One example study is the Generative Aspect-oriented Testing framEwork (GATE) framework, proposed by Feng et al. to test NFRs, e.g. performance, reliability, dependability and programming standard enforcement, in component based systems [38]. The GATE framework contains an expandable repository of reusable aspect test cases that are used to generate Executable Aspect Test Cases (EATC) for unit testing. The generated test cases are then weaved into the component source code as aspects by a post-weaving tool. The weaving is followed by execution of the test cases through the framework’s test engine that also generates a test report. In other example studies aspects are used to extend unit tests in the Java environment [56-57]. The aspects are also used on a meta-level to monitor unit test execution, i.e. not in the actual test cases themselves, but rather to measure the test efficiency or test coverage of the unit tests [58].

Besides aspect orientation, NFR unit testing of components can also be performed by other means. One of these methods is proposed by Hill et al. with their tool called Understanding Non-functional Intentions via Testing and Experimentation (UNITE). UNITE is developed explicitly for NFRs unit testing of component-based distributed systems, based on two key concepts [10]. The concepts are relational database theory and text based system logs generated from a testing environment that are used to capture and store software metrics. The relational database consists of a data table that is used to store associated metrics for non-functional unit test evaluation. The system logs in turn contain a set of variables (tags) that capture metrics of interest. Testers are able to identify the metrics of interest within the log messages using message constructs. In other words, they specify data messages of interest by using placeholders to tag data or variables that are then extracted from the system log. Hence, by formulating equations based on the captured metrics, testers are able to define unit tests for NFR testing.

Some prior research on NFR testing has focused on modern enterprise software systems [39]. The reason is because software system environments are growing in complexity that makes them increasingly difficult to accurately and realistically represent for testing. The combined factors of a complex operational environment and the often critical nature of the task being carried out by the system make rigorous testing of enterprise software systems extremely important [39]. Tools exist to test enterprise systems performance, e.g. SLAMD Distributed Load Generation Engine, HP’s Load Runner, etc. These tools enable software testers to simulate many thousands of

enterprise system clients using minimal resources. However, these tools have been designed to only generate and cover scalable client load against a server system under test [39]. Hence, they do not cover generation of server responses to client load, and server-to-server communications, which are required to test enterprise software. Some of these tools are expensive and restricted in terms of license, proprietary features, and configuration [4]. They are also seldom portable and are therefore hard to share on an ongoing basis [4]. Each of these tools includes a proprietary scripting language that requires training to be understood [4]. In order to overcome these problems Hine et al. propose the use of a reactive emulation environment that replaces an enterprise system's deployment environment for non-functional testing [39]. The main idea is to model the run-time behavior of each system (which is called an endpoint) in the environment based on the protocol specifications of the expected interactions. Hence, each endpoint is replaced by an instantiation of the endpoint's corresponding model in the emulation environment. If the endpoint model is successful in mimicking the interaction behavior of the actual system then it is considered scalable. Stankovic proposes another solution based on an in-house distributed framework for performance testing [4]. The framework is based on Visper [59] that is an object-oriented distributed programming middleware. The choice of Visper as the basis for the framework is supported by the characteristics of the Visper model. Visper is adaptable for rapid application development and produces results of high quality.

Benefits

- Aspect oriented techniques promote modularized test code that in turn simplifies maintenance after requirements change due to design trade-offs [7]. Compared to conventional modularization techniques, they provide more explicit alignment between requirements and test code that improves traceability [7]. Additionally, aspect oriented code can be handled by the same development tools and environments as conventional code and therefore does not add any overhead [7].
- The GATE framework proposed in [38] improves reusability and automatic creation of aspect oriented test cases with proven benefits to software quality.
- The component aspects technique proposed in [37] has been shown to be applicable for automation of validation tests of deployed software components. Additionally, compared to many other approaches which require manual effort, this technique is autonomous, using "validation agents" to proactively carry out tests.
- UNITE provides a simple way for unit testing of NFRs during the early stages of the software development lifecycle [10]. Data tables in UNITE constitute a repository of historical data that can be used for further analysis and monitoring during development [10]. Additionally, the definition of NFR unit tests becomes less complex, as the only task required for evaluating a new NFR is to add new messages to the system log to capture metrics.
- The reactive emulation environment proposed in [39] is a valuable approach for NFR testing in modern enterprise software systems. The emulated environment allows software developers to conduct testing of the enterprise software in a realistic deployment

environment without the need to physically build it. This is of great benefit since these environments are often impractical, or even impossible, to construct [39].

- The in-house framework for performance testing proposed in [4] is said to have high usability, availability, modifiability, and scalability [4].

Limitations

- One of the generic problems with the use of aspect oriented techniques for testing component based systems is code manipulation [7]. In order to create a test harness, one has to weave test aspects into the system. This in turn requires extra development steps and specific versions of the system to be built that adds complexity to the testing process [7]. Performance testing using aspect orientation techniques is also problematic because it adds overhead that affects the actual performance of the system [7]. Lack of tool support is another limitation that has harmed the industrial adoption of aspect oriented testing. Lack of tool support also makes it difficult to define testing practices that can be integrated in existing processes and tool chains [7].
- The "component aspects" method proposed in [37] also suffers from some limitations:
    - It is difficult to identify the reason why a deployed component does not comply to its constraints.
    - Handling multiple aspect interactions is difficult.
    - Additional work is required to specify the detailed aspect information needed for each component implementation.
- The reactive emulation environment proposed in [39] is unable to modify operations in the emulated endpoint data pool; a limitation in all enterprise software system components that need to manipulate data on endpoints.

Discussion

- Figure 2 shows that most NFR testing research is focused on MBT, (10/24-42%). In MBT, models are used to represent and resolve domain specific problems through automatic generation and execution of test cases and test data. Examples of systems that can be tested using MBT include: distributed systems, telecommunication systems, service based software systems, and software intensive networked automation systems.
- Models can also be used to derive TTCN-3 test cases for non-functional properties such as timing properties. TTCN-3 has gained industrial appeal as a successful test specification language and future research on combining MBT and TTCN-3 should therefore be conducted.
- Many NFR testing techniques for CBSE suffer from problems such as scattering and tangling component concerns within the design. These problems can be resolved by using aspect-oriented techniques. These techniques could also be used for NFR testing. Further research into this technique should therefore be conducted in the future.
- The review has also uncovered a set of problems related to NFR testing of web services. These weaknesses include different naming conventions of quality attributes for web

services and a limited amount of studies on the subject. Further research on NFR testing for web services is therefore required.

- According to Figure 3, (12/24-50%) of the studies are published in the last three years (2008-2010), hence a positive growth of studies into relevant testing techniques for NFRs.

3.2. RQ2. What type of solutions is represented in the studies on NFR testing?

Figure 4 shows that most studies on the NFR testing are of the type "Solution Proposal" (13/24 - 54%) or "Evaluation Research" (8/24 - 33%). Figure 5 shows the distribution of contribution type over the publications. It should be noted that a publication might provide multiple contributions e.g. both a tool and method. The contributions in the publications are primarily "method" (12/24 - 50%) and "frameworks" (7/24 -29%). Figure 8 also shows that the primary contribution type in all research focuses, except from "standard and languages", has been "method". A considerable amount of the studies (16/24 – 67%) are directed at specific domains, as shown in Figure 6. The majority of the publications are academic research studies (19/24 -79%), as given in Figure 7.

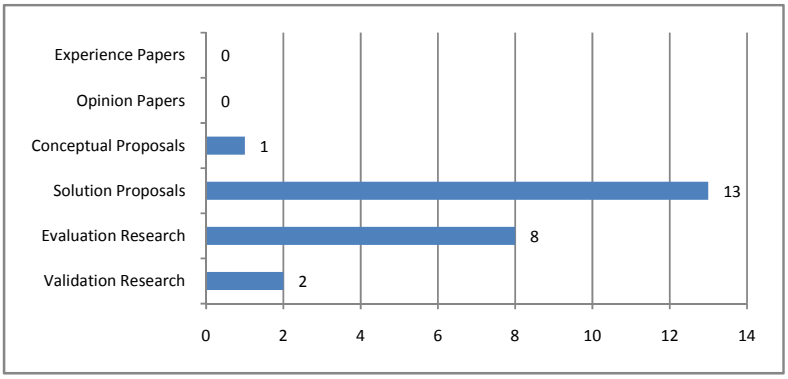

Figure 4. The distribution of research type – NFR testing

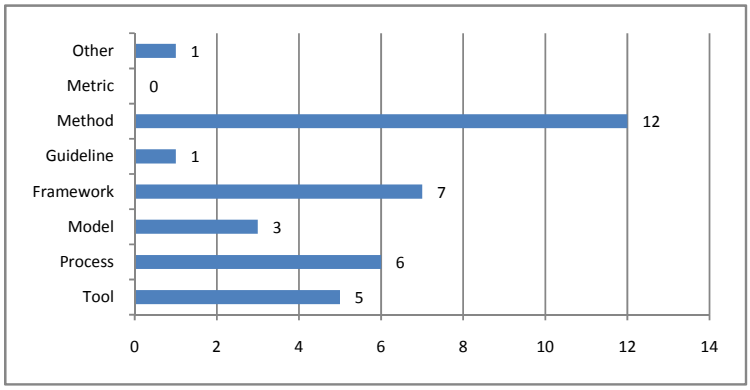

Figure 5. The distribution of contribution type – NFR testing

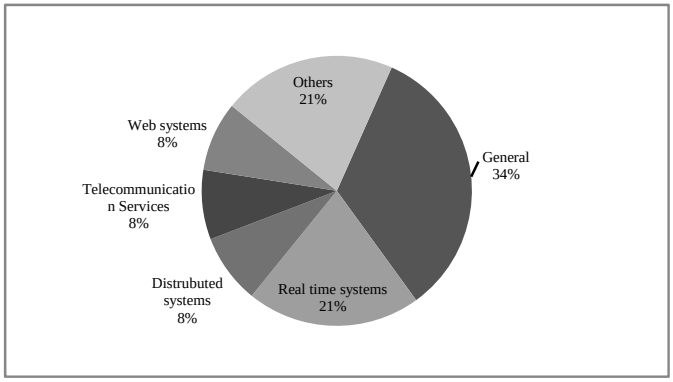

Figure 6. The distribution of publication domain

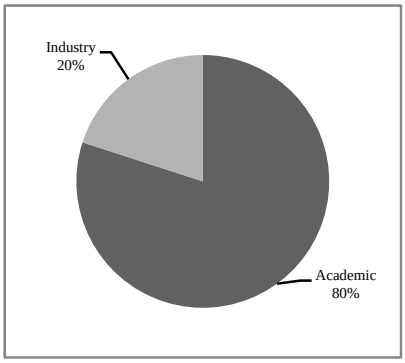

Figure 7. The distribution of publication context

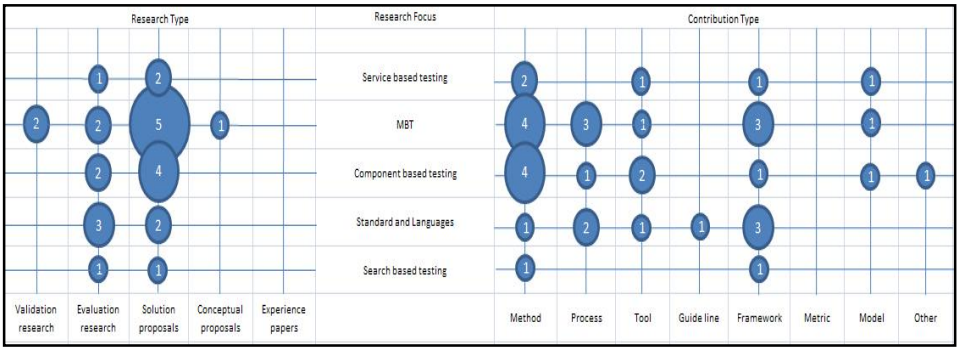

Figure 8. Map of research focus - NFR testing

Discussion

- Most of the publications that were uncovered during the review are "Solution Proposal". Hence, very little work has been conducted on solution evaluation. In other words, the challenges for each research area are well understood but there is still a gap in the current body of knowledge regarding validated solutions to these problems. In cases where a solution has been validated it has been conducted by the researcher him/her self and, not

by other independent researchers, lowering the confidence of the solution. Hence, further work must be done to validate the proposed solutions both in academia and industry.

- The number of papers of the "Evaluation Research" type suggests that considerable effort has been put into the investigation of NFR testing or implementation testing techniques in practice in recent years.
- It can be inferred from Figure 5 and Figure 8 that the emphasis of prior research on NFR testing is on proposing new methodologies or enhancing existing ones for NFR testing. However, as methods need supportive frameworks and tools in order to be used in practice; further effort is required in these areas.
- Most of the found publications are academic. Hence, there is a need to verify these methods in industry to show their applicability. However, continued research into methods that have already been proven as industrially applicable is also required.
- A considerable percentage of the found studies were specific to one or more particular domain(s) that means the adaptability of the proposed solutions in other industrial domains have to be investigated. Further results that show the solutions work in more domains would hence be a step towards the industrialization of these approaches.

4. Conclusion

This paper presents a systematic review of the body of knowledge on NFR testing. The systematic review approach was chosen because it provides a more comprehensive result in terms of coverage, whilst being more repeatable, than a conventional and informal survey. These properties are achieved through the use of well-defined paper selection and analysis procedures [20]. Below the results of the systematic review have been summarized:

- Most NFR testing research is focused on MBT, followed by component based testing. NFRs play a significant role in the success of a software project. Therefore, all software development approaches should provide practical solutions for efficient testing of NFRs. According to the result of this review this need is understood to an extent in the MBT approach. However, even with this approach the overall number of solutions (10 solutions in the last 10 years) is by no means enough to fulfill the industrial needs.
- There are various proposed solutions that address the challenges in NFR testing. However, most solution proposals do not fully evaluate the proposed techniques and there is therefore a need for further studies that focus on proposal validation.
- The contribution types of most of the found publications were either "method" or "framework". This result encourages further research into new methodologies, improvement of existing ones, and development of support frameworks and tools to make the methods applicable in industry.
- Studies on NFR testing have been primarily conducted in academia. Further effort is therefore required to verify the industrial applicability of the study results.
- There is also a need for publications where the proposed methods for NFR testing are compared. Comparative studies will provide industry with information regarding what

method best fits their needs and will also help researchers to find the advantages and disadvantages of their methods that require further work.

5. References

1. Grossmann J, Serbanescu D, Schieferdecker I (2009) Testing Embedded Real Time Systems with TTCN-3. Paper presented at the Proceedings of the 2009 International Conference on Software Testing Verification and Validation,

2. Dulz W, Fenhua Z MaTeLo - statistical usage testing by annotated sequence diagrams, Markov chains and TTCN-3. In: Third International Conference on Quality Software, 2003. Proceedings. , 6-7 Nov. 2003 2003. pp 336-342

3. Saifan A, Dingel J (2010) A Survey of Using Model-Based Testing to Improve Quality Attributes in Distributed Systems. In: Elleithy K (ed) Advanced Techniques in Computing Sciences and Software Engineering. Springer Netherlands, pp 283-288. doi:10.1007/978-90-481-3660-5_48

4. Stankovic N Patterns and Tools for Performance Testing. In: Electro/information Technology, 2006 IEEE International Conference on, 7-10 May 2006 2006. pp 152-157

5. Grunske L Specification patterns for probabilistic quality properties. In: Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on, 10-18 May 2008 2008. pp 31-40

6. Matoussi A, Laleau Re (2008) A Survey of Non-Functional Requirements in Software Development Process. Technical Report

7. Metsa J, Katara M, Mikkonen T Testing Non-Functional Requirements with Aspects: An Industrial Case Study. In: Quality Software, 2007. QSIC '07. Seventh International Conference on, 11-12 Oct. 2007 2007. pp 5-14

8. Lindstrom DR (1993) Five Ways to Destroy a Development Project. IEEE Software 10 (5):55 - 58

9. Doerr J, Kerkow D, Koenig T, Olsson T, Suzuki T, Society IC Non-functional requirements in industry - Three case studies adopting an experience-based NFR method. In: 13th IEEE International Conference on Requirements Engineering, Proceedings, 2005. pp 373-382

10. Hill JH, Turner HA, Edmondson JR, Schmidt DC (2009) Unit Testing Non-functional Concerns of Component-based Distributed Systems. Paper presented at the Proceedings of the 2009 International Conference on Software Testing Verification and Validation,

11. Chung L, do Prado Leite J (2009) On Non-Functional Requirements in Software Engineering. In: Borgida A, Chaudhri V, Giorgini P, Yu E (eds) Conceptual Modeling: Foundations and Applications, vol 5600. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp 363-379. doi:10.1007/978-3-642-02463-4_19

12. Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE-2007- 01

13. Wieringa R, Maiden N, Mead N, Rolland C (2005) Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir Eng 11 (1):102-107. doi:10.1007/s00766-005-0021-6

14. Cai KY, Card D (2006) An analysis of research topics in software engineering. Journal of Systems and Software 81

15. Herrmann A, Kerkow D, Doerr J Exploring the characteristics of NFR methods - a dialogue about two approaches. In: Requirements Engineering: Foundation for Software Quality. 13th International Working Conference, REFSQ 2007, 11-12 June 2007, Berlin, Germany, 2007. Requirements Engineering: Foundation for Software Quality. Proceedings 13th International Working Conference, REFSQ 2007. Springer-Verlag, pp 320-334

16. Chung L, Nixon B, Yu E, Mylopoulos J (1999) Non- Functional Requirements in Software Engineering. Kluwer Academic Publishers

17. Chirinos L, Losavio F, Matteo A (2004) Identifying quality-based requirements. Information Systems Management 21 (1):15-26

18. Boegh J (2008) A new standard for quality requirements. IEEE Software 25 (Copyright 2008, The Institution of Engineering and Technology):57-63

19. Sindhgatta R, Thonse S (2005) Functional and non-functional requirements specification for enterprise applications. In: Bomarius F, KomiSirvio S (eds) Product Focused Software Process Improvement, Proceedings, vol 3547. Lecture Notes in Computer Science. pp 189-201

20. Shahrokni A, Feldt R Towards a Framework for Specifying Software Robustness Requirements Based on Patterns. In: Requirements Engineering: Foundation for Software Quality. 16th International Working Conference, REFSQ 2010, 30 June-2 July 2010, Berlin, Germany, 2010. Requirements Engineering: Foundation for Software Quality. Proceedings 16th International Working Conference, REFSQ 2010. Springer, pp 79-84

21. Onabajo A, Jahnke JH Modelling and reasoning for confidentiality requirements in software development. In: Riebisch M, Tabeling P, Zorn W (eds) 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Proceedings - MASTERING THE COMPLEXITY OF COMPUTER-BASED SYSTEMS, 2006. pp 460-467

22. Glinz M On Non-Functional Requirements. In: Requirements Engineering Conference, 2007. RE '07. 15th IEEE International, 15-19 Oct. 2007 2007. pp 21-26

23. Ivarsson M, Gorschek T (2009) Technology transfer decision support in requirements engineering research: a systematic review of REj. Requirements Engineering, vol 14, 2009, pp 155-175

24. Glass RL, Vessey I (1995) Contemporary Application Domain Taxonomies. IEEE Softw

25. Freelon D ReCal: reliability calculation for the masses. http://dfreelon.org/utils/recalfront/.

26. Hanna S, Munro M Fault-Based Web Services Testing. In: Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on, 7-9 April 2008 2008. pp 471-476

27. Yeom G, Yun T, Min D (2006) QoS Model and Testing Mechanism for Quality-driven Web Services Selection. Paper presented at the Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06),

28. Hanna S, AbuAli A (2010) Platform Effect on Web Services Robustness Testing.

29. Dulz W (2010) On-the-Fly Testing by Using an Executable TTCN-3 Markov Chain Usage Model. In: Maciaszek LA, González-Pérez C, Jablonski S (eds) Evaluation of Novel Approaches to Software Engineering, vol 69. Communications in Computer and Information Science. Springer Berlin Heidelberg, pp 17-30. doi:10.1007/978-3-642-14819-4_2

30. Lencevicius R, Metz E, Ran A (2002) Software validation using power profiles. Paper presented at the Proceedings of the 24th International Conference on Software Engineering, Orlando, Florida,

31. Correia A, Abreu FBe (2010) Defining and Observing the Compliance of Service Level Agreements: A Model Driven Approach. Paper presented at the Proceedings of the 2010 Seventh International Conference on the Quality of Information and Communications Technology,

32. Hussain T, Eschbach R Automated fault tree generation and risk-based testing of networked automation systems. In: Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on, 13-16 Sept. 2010 2010. pp 1-8

33. Corriveau J-P (2007) Testable requirements for offshore outsourcing. Paper presented at the Proceedings of the 1st international conference on Software engineering approaches for offshore and outsourced development, Zurich, Switzerland,

34. Gregoriades A, Sutcliffe A (2005) Scenario-Based Assessment of Nonfunctional Requirements. IEEE Trans Softw Eng 31 (5):392-409. doi:10.1109/tse.2005.59

35. Jaskó S, Dulai T, Muhi D, Tarnay K Test aspect of requirement specification.

36. Alexander I (2003) Misuse cases: use cases with hostile intent. Software, IEEE 20 (1):58-66

37. Grundy J, Ding G, Hosking J (2005) Deployed software component testing using dynamic validation agents. J Syst Softw 74 (1):5-14. doi:10.1016/j.jss.2003.05.001

38. Yankui F, Xiaodong L, Kerridge J A product line based aspect-oriented generative unit testing approach to building quality components. In: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, 24-27 July 2007 2007. pp 403-408

39. Hine C, Schneider J-G, Han J, Versteeg S (2009) Scalable Emulation of Enterprise Systems. Paper presented at the Proceedings of the 2009 Australian Software Engineering Conference,

40. Deiss T, Nyberg AJ, Schulz S, Teittinen R, Willcock C (2006) Industrial Deployment of the TTCN-3 Testing Technology. IEEE Softw 23 (4):48-54. doi:10.1109/ms.2006.108

41. IEEE Standard. Adoption of International Standard ISO/IEC 12119: 1994(E). Information Technology - Software Packages - Quality Requirements and Testing (2007).

42. Ghani K, Clark JA (2009) Automatic Test Data Generation for Multiple Condition and MCDC Coverage. Paper presented at the Proceedings of the 2009 Fourth International Conference on Software Engineering Advances,

43. Afzal W, Torkar R, Feldt R (2009) A systematic review of search-based testing for non-functional system properties. Inf Softw Technol 51 (6):957-976. doi:10.1016/j.infsof.2008.12.005

44. Lazowska E, Zahorjan J, Graham G, Sevcik K (1984) Quantitative System Performance: Computer System Analysis Using Queuing Network Models. Prentice-Hall

45. Bernardi S, Donatelli S, Jos\, \#233, Merseguer (2002) From UML sequence diagrams and statecharts to analysable petri net models. Paper presented at the Proceedings of the 3rd international workshop on Software and performance, Rome, Italy,

46. R.Pooley (1999) Using UML to Derive Stochastic Process Algebra Models. In Proceedings of the 15th UK Performance Engineering Workshop

47. Jürjens J (2005) Secure Systems Development With UML. SpringerVerlag

48. Jürjens J Towards Development of Secure Systems Using UMLsec. In: FASE '01 Proceedings of the 4th International Conference on Fundamental Approaches to Software Engineering 2001. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp 187-200. doi:10.1007/3-540-45314-8_14

49. Veanes M, Campbell C, Grieskamp W, Schulte W, Tillmann N, Nachmanson L (2008) Model-based testing of object-oriented reactive systems with spec explorer. In: Robert MH, Jonathan PB, Mark H (eds) Formal methods and testing. Springer-Verlag, pp 39-76

50. Tretmans G, Brinksma H, Resyste Cd (2002) Automated Model Based Testing. In Proc of the 3rd PROGRESS workshop on Embedded Systems

51. Herrmann A, Paech B (2008) MOQARE: misuse-oriented quality requirements engineering. Requirements Engineering 13 (1):73-86. doi:10.1007/s00766-007-0058-9

52. Jaskó S, Dulai T, Muhi D, Tarnay K (2010) Test aspect of requirement specification. Comput Stand Interfaces 32 (1-2):1-9. doi:10.1016/j.csi.2008.12.005

53. Amirat A, Laskri M, Khammaci T (2008) Modularization of crosscutting concerns in requirements engineering. International Arab Journal of Information Technology 5 (2):120-125

54. Hussein M, Zulkernine M UMLintr: A UNIL profile for specifying intrusions. In: Riebisch M, Tabeling P, Zorn W (eds) 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Proceedings - MASTERING THE COMPLEXITY OF COMPUTER-BASED SYSTEMS, 2006. pp 279-286

55. Dal Cin M Structured language for specifications of quantitative requirements. In: Proceedings of HASE 2000. 5th IEEE International Symposium on High-Assurance Systems Engineering, 15-17 Nov. 2000, Los Alamitos, CA, USA, 2000. Proceedings. Fifth IEEE International Symposium on High Assurance Systems Engineering (HASE 2000). IEEE Comput. Soc, pp 221-227

56. Stamey J, Saunders B (2005) Unit testing and debugging with aspects. J Comput Small Coll 20 (5):47-55

57. Uir\, \#225, Kulesza, Cl\, Sant'Anna u, Lucena C (2005) Refactoring the JUnit framework using aspect-oriented programming. Paper presented at the Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, San Diego, CA, USA,

58. Rajan H, Sullivan K (2005) Aspect language features for concern coverage profiling. Paper presented at the Proceedings of the 4th international conference on Aspect-oriented software development, Chicago, Illinois,

59. Stankovic N, Zhang K (2002) A Distributed Parallel Programming Framework. IEEE Transactions on Software Engineering 28 (5):478-493

2011 Fourth International Conference on Software Testing, Verification and Validation Workshops

Alignment of requirements specification and testing: A systematic mapping study

Zeinab Alizadeh Barmi
barmi@student.chalmers.se

Amir Hossein Ebrahimi
amirho@student.chalmers.se

Robert Feldt
robert.feldt@chalmers.se

Department of Computer Science and Engineering
Chalmers and Gothenburg University
Goteborg, Sweden

Abstract

Requirements should specify expectations on a software system and testing should ensure these expectations are met. Thus, to enable high product quality and efficient development it is crucial that requirements and testing activities and information are aligned. A lot of research has been done in the respective fields Requirements Engineering and Testing but there is a lack of summaries of the current state of the art on how to link the two. This study presents a systematic mapping of the alignment of specification and testing of functional or non-functional requirements in order to identify useful approaches and needs for future research. In particular we focus on results relevant for non-functional requirements but since only a few studies was found on alignment in total we also cover the ones on functional requirements. The map summarizes the 35 relevant papers found and discuss them within six major sub categories with model-based testing and traceability being the ones with most prior results..

1. Introduction

With the ever-growing market demand for high quality software, the need and importance of testing has become more apparent in recent years. For more safety critical software systems - like medical diagnosis and space shuttle missions - software testing is a crucial aspect of their success since software errors can cause irreparable losses. On the other hand, Requirements Engineering (RE) represents a complementary view of a system and thus has a synergistic relationship with testing [1]. Therefore bringing RE and testing closer could benefit both disciplines. Making a strong link between them will improve the outcome of the software development process [2]. It also helps to discover possible errors early, which in turn will improve the product quality and lead to more satisfied customers [3]. From the project management perspective, linking requirements and testing would help to reach a more accurate testing plan which in turn would improve project cost and schedule estimation. The likely result for the project is to be finished within the planned schedule and budget [1]. Although organizations are becoming more interested in linking requirements and testing, but often this link is not provided and there is a gap between them. It is noticeable that in these efforts, the focus has been mainly on FRs rather than NFRs. NFRs play a significant role in the success of software projects. Grunske[4] states that NFR's fulfillment is often more important than implementing FRs to have a satisfied customer. Matoussi and Laleau [5] point out that verification of NFRs are almost always done very late after finishing the implementation. Our aim in this research is to perform a systematic mapping on the alignment of functional or non-functional requirement specification and testing to get an overview of existing research in this area.

Among the work which has been done on alignment a lot of attention has been given to traceability. Traceability of requirements can help determine what requirement has been covered by which test and how the generated test cases cover these requirements [6]. Tracing from tests back to requirements is also helpful to find the root of a failed test. Another important reason for traceability is improving change management by helping to find out how change in the requirement is reflected in the test cases. Also in the alignment research area model based development has attracted a lot of attention. The idea behind MBT is the derivation of executable test code from test models by analogy to Model Driven Architecture (MDA) [6]. This technique is becoming of more interest in industry because it provides automatic deriving of test cases

978-0-7695-4345-1/11 $26.00 © 2011 IEEE
DOI 10.1109/ICSTW.2011.58

from the behavioral model of the system called the test model.

As Petersen et al. [7] describe, a systematic mapping consists of an overview of primary studies performed on a specific topic and the categorization of the results by providing a visual summary. As such, a systematic map offers an overview of a field, identifying potential gaps in research, whereas a systematic literature review [8]provides a more detailed study of the identified results. The systematic mapping process consists of five phases [7]: 1) Definition of the research questions, 2) Conducting the search for primary studies, 3) Screening of papers for inclusion or exclusion, 4) Keywording the abstracts, and 5) Data extraction and mapping of the studies.

This paper is organized as follows: Section 2 describes the phases of our systematic mapping process. Some of these phases are broken down into smaller steps. In section 3 the answer to our research questions is provided. Discussion and conclusions are provided in sections 4 and 5, respectively.

2. Research Method

2.1. Definition of research questions

To investigate existing research on the alignment of functional or non-functional requirement specification and testing, we formulated the following research questions:

RQ1. Which studies have been done on linking the specification and testing of requirements?

Aim: We want to find out which topics have been investigated and to what extent? Which of these studies are focused on NFRs? What are the different perspectives that address the alignment of requirements and testing, e.g., how common testing approaches try to address alignment or traceability? This would help identify the needs for complementary research.

RQ2. What types of solutions are represented in these researches? We want to find the solutions given in different topics that address alignment such as method, process, framework, tool, etc.

RQ3. In which fora is research on alignment of requirement and testing published? An overview of the range of fora in which the researches are published also could help with our research.

2.2. Conducting the search for primary studies

In order to conduct our search string we needed to obtain an overview of the requirements specification and testing area and the alignment of these two. So we gathered an initial set of previously known publications in the area, through an exploratory search [1, 3-5, 9-14]. We then tried to extend our initial set using forward/backward referencing, i.e. looking at which papers were referenced in or referred to papers in our initial set. The study of the resulting 24 papers helped us to explore and choose relevant keywords for the systematic search. From our research questions and based on our study of the initial set of papers we derived some categories for conducting the search string. These categories focused on NFRs (named C1), specification of NFRs (named C2), testing of NFRs (named C3), and linking the specification and testing of NFRs (named C4). We formulated a combination of these categories to reach our search string, which was "C1 AND C2 AND C3 AND C4". As we decided to cover all researches in the last 10 years, we limited the search on papers that were written in English and published between 2001 and 2010 (It should be noted that since the search was done in November of 2010 not all 2010 papers will be included in the results). We conducted the search on 4 databases (Scopus, Inspec Engineering Village, ISI Web of Knowledge, and IEEE Xplore). We then followed an incremental refinement of the search string in five steps. In each refinement step we checked the results to see if they contained papers from our initial set that were on the alignment area. In case the items of the initial set were missed or the results were not relevant we improved our search string further. One major refinement was removing the C2 (specification) category from the search string. The reason was that many papers in our initial set did not include the terms belonging to the C2 (specification) category in their title or abstract. As there was little research with focus on NFRs we decided to get some ideas from the alignment of requirements and testing in general. So we added another category named C5 with the "requirement" item and we changed our search string to "(C1 OR C5) AND C3 AND C4". The final version of categories is shown in Table 1. After the final iteration of the search string refinement we reached 591 hits.

Table 1. Search string categories

NFR (C1)
"nonfunctional requirement" OR "nonfunctional requirements" OR "non functional requirement" OR "non functional requirements" OR "non functional software requirement" OR "non functional software requirements" OR "nonbehavioral requirement" OR "nonbehavioral requirements" OR "nonbehavioural requirement" OR "nonbehavioural requirements" OR "non behavioral requirement" OR "non behavioral requirements" OR "non behavioural requirement" OR "non behavioural requirements" OR "nonfunctional property" OR "nonfunctional properties" OR "non functional property" OR "non functional properties" OR "quality attribute" OR "quality attributes" OR "quality requirement" OR "quality requirements" OR "quality attribute requirement" OR

"quality attribute requirements"
Testing (C3)
"test" OR "testing" OR "verify" OR "verifying" OR "verification" OR "validate" OR "validating" OR "validation"
Alignment (C4)
"align" OR "aligning" OR "alignment" OR "trace" OR "tracing" OR "traceable" OR "traceability" OR "link" OR "linking" OR "bridge"
Requirement (C5)
Requirement

2.3. Screening papers for inclusion or exclusion and keywording the abstracts

In this phase we defined inclusion and exclusion criteria in order to achieve a common understanding between the team members that would perform the screening. The paper screening process was performed in two steps.

In the first step the title and abstract (if needed) of all papers were considered. The main criteria for inclusion were papers that describe alignment of specification and testing of functional or non-functional requirements. Conference proceedings and papers that were not focused on software development, for example papers that focused on hardware or network development were excluded. In this step if a researcher was unsure about excluding a paper, this paper was included for the second step. 546 papers were excluded during this step. In the second step the full text of the papers were studied. Posters, opinion papers i.e. papers that express the personal opinion of author on what is good or bad [15], and short papers (with less than 6 pages) were excluded. Papers were only included if they had been subject to peer review. If researchers did not agree on the inclusion or exclusion of some papers these papers were discussed until a decision was made by consensus. 10 papers were excluded during this step. At the end the number of hits was reduced to 35papers.

During this phase we developed our data extraction form, partly based on the one used by Ivarsson and Gorschek[16]. To build this form some keywords and concepts, like context and domain were reached through the study of paper abstracts by each researcher. The keywords were evolved as papers were studied in detail. Using the keywords, we finally came up with the following key attributes in the form *research focus* (Model-centric approaches, code-centric approaches/ traceability/ formal approaches/ test cases/ problems and set of good practices), *contribution type* (Tool/ process/ model, framework/ guideline/ method/ metric and other), *quality requirements/attributes the paper focus on*, *research method*, *context* (academia/industry/ open-source software), *domain*, and *scale*. It should be noted that some research focus items contain some subcategories. Model centric approaches include 2 subcategories of Model based testing (MBT) and Goal-oriented development. Code centric approaches category is divided into 3 subcategories of Test driven development (TDD), Storytest driven development and Behavior driven development (BDD). Test cases category is also broken down to 2 subcategories of Test case generation (manual/automatic) and Test case coverage. During the study of the full text of included papers this extraction form was filled for each individual paper.

3. Results (data extraction and mapping the studies)

As the full text of the papers was being studied the data extraction form for each paper was also filled. We answer our research questions by analyzing the extracted data from the papers.

3.1. RQ1. Which studies have been done on linking the specification and testing of requirements?

We have identified several research focuses on the alignment of requirements specification and testing. In the last part of this section we have mentioned which of these approaches support NFRs. The distribution of research focus is shown in Figure 1.The distribution of research focus over different years is shown in Table 2.

3.1.1. Model Based Testing (MBT)

In Model based testing (MBT) which has the highest focus, informal requirements of the system are the base for developing a test model which is a behavioral model of the system. This test model is used to automatically generate test cases. One problem in this area is that the generated tests from the model cannot be executed directly against an implementation under test (IUT) because they are at the same level of abstraction as the model. Arnold et al. address this problem [17] and propose a solution. They also claim that their scenario-driven approach supports the traceability between generated and executed test cases, and executions of an IUT. Their approach supports both FRs and NFRs.

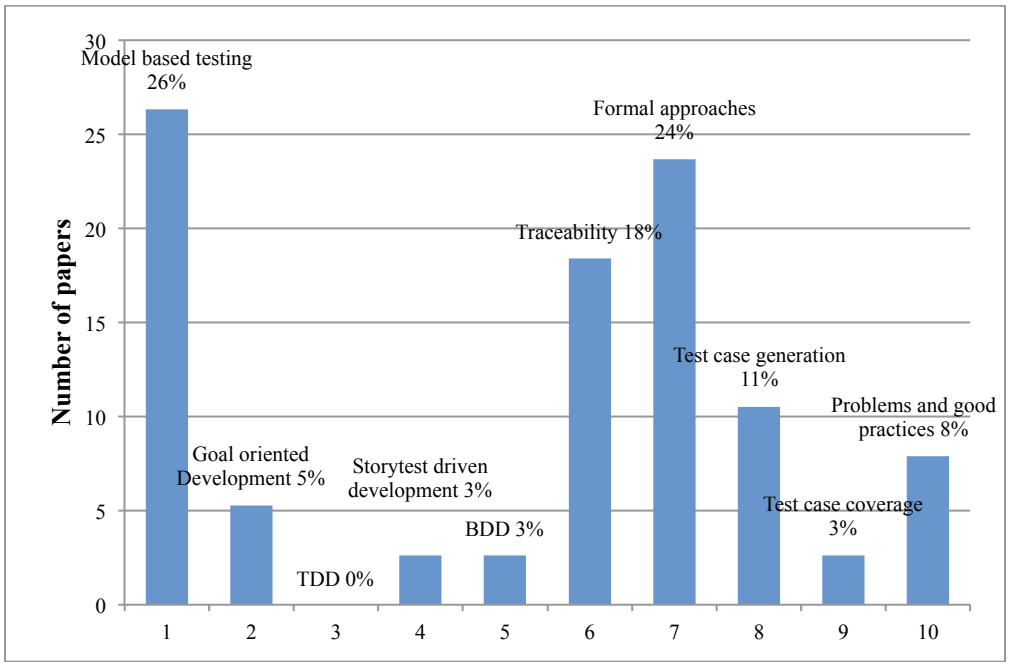

Figure 1. Distribution of research focus

There are some other prior researches that are scenario based. Goel et al. [18] propose a model-driven approach in which strengths of both scenarios-based and state-based modeling styles are combined. Their Footprinter tool makes it possible to trace from requirements to testing and vice versa in a round-trip engineering approach. The Web Services Testing Framework (WSTF) is proposed by Tasi et al. for Scenario-based testing of Web Services [19]. In this approach test scripts and test cases are automatically generated based on a scenario specification by the test master component. Some prior research address development process in model based testing. Pfaller et al. propose [20] using different levels of abstraction in the development process to derive test cases and link them with the corresponding user requirements. Boulanger and Dao propose an approach [21] in which RE is done in different phases of the V-model in order to facilitate requirements validation and traceability. Lobo and Arther have worked on a project [22] which aims to reduce the gap between RE and the V&V process. This is done by applying V&V in a two-phase model: in the first phase, V&V is performed right after requirements elicitation for each distinct function of the system. In the second phase, the quality of linkages between requirement sets is verified and validated. Several studies are on model based testing of service oriented systems. Felderer et al. focus on model–driven testing of service-oriented systems in a test–driven manner [6]. They believe that Telling TestStories tool could support traceability between all kinds of modeling and system artifacts. Tasi et al. proposed framework - which was described as a scenario based approaches also aims for testing of web services [18]. There are also other MBT approaches. Marelly et al. extend sequence charts (LSCs) with symbolic instances and symbolic variables [23] in order to reach linking requirements and testing.

Zou and Pavlovski propose control cases to complement use cases by modeling NFRs which cannot be addressed by use cases [11]. They can be applied during the software development life cycle and can also be used to verify whether the implemented system meets the specified NFRs.

3.1.2. Goal oriented development

In the field of Model driven engineering (MDE), Goal-oriented modeling can easily realize stakeholder's concerns and their interdependencies using concepts which much less dependent on the underlying implementation technology and much closer to the problem domain [24].

Goals, which can be at various levels of abstraction, define stakeholders' expectations from the system [24]. Hard goals are states that actors can attain and soft goals are goals which can never be fully satisfied. Both FRs and NFRs can be represented by goal oriented modeling. FRs are shown by hard goals, and NFRs like efficiency and reliability are represented by soft goals which means they are expected to be satisfied within an acceptable limits rather than absolutely [24].

Table 2. Distribution of research focus over years

Research Focus	2002	2003	2004	2005	2006	2007	2008	2009	2010	Total	%
Model based testing				1	3		1	2	3	10	26
Goal- oriented development					1				1	2	5
Storytest driven development				1						1	3
BDD									1	1	3
Traceability				1	1		3	1	1	7	18
Formal Approaches		1				1	2	1	4	9	24
Test case generation				1	1		1	1		4	11
Test case coverage					1					1	3
Problems and good practices				2					1	3	8
Total	0	1	0	6	7	1	7	5	11	38	

This type of development improves productivity as well as model quality. Nan et al. propose a framework [24] for tracing aspects from requirement to implementation and testing. Language support is also provided to transform models into aspect oriented programs. Test cases can be derived from these models to help in the verification process [24].

3.1.2.1. Agent oriented Software Engineering (AOSE)

AOSE methodologies are based upon the Agent paradigm, and help to develop complex distributed systems [25]. AOSE partially addresses the link between requirements and testing. This is done by specification-based formal verification and object oriented testing techniques. Duy et al. introduce a testing framework for AOSE methodologies called TROPOS [25]. This framework provides a systematic way of deriving test cases from goal analysis. This approach is called goal oriented testing.

3.1.2.2. Aspect Oriented Software Development (AOSD)

Nan et al. present AOSD as a solution for transformation from the goal model to the implementation [24]. They describe how aspects are introduced from the goal models and introduce a framework with which aspects can maintain traceability from the requirement level down to implementation and test levels.

3.1.3. Traceability

Abbors et al. [26] present an approach for requirements traceability across a MBT process and the tools that are used for each phase. Some prior researches address requirement based testing to facilitate traceability between requirements and testing. Méndez et al. present a requirement-based testing process and define a guideline on how to keep the traceability among requirements down to the test cases in this process [27]. Quinn et al. propose the TraceFunction Method by which requirements of a software component can be specified in one easily used reference document in order to facilitate traceability between requirements and testing [28]. Some researches address traceability issues using scenarios. Naslavsky et al. believe that different kinds of scenarios are useful for tracing requirements to tests through the development life cycle and each can be used as test scenarios [29]. Test generating and traceability issues in three different scenario-based system modeling languages are studied by Goel and Roychoudhury [30]. Other researches construct meta-models to facilitate traceability. In order to establish the relationship between software components that include the requirements, design test cases and code, Ibrahim et al. construct a meta-model with top-down and bottom-up traceability support [31].Dubois et al. propose a meta-model called DARWIN4REQ which aims to keep the traceability link between three phases of requirement elicitation, design and V&V of requirements [32].

3.1.4. Formal approaches

Post et al. focus on translating requirements into scenario-based formal language which in turn could be linked to software verification [2]. Bouquet et al. use a

subset of UML 2.0 diagrams and Object Constraint Language (OCL) operators to formalize the expected system behavior [33]. The model is used for automatically generating executable test scripts. Kelleher and Simonss propose a new requirement modeling approach [34] in which use cases are replaced with use-case classes in UML 2.0. Use case classes are formal templates for describing rules on modeling requirements with instances. This replacement, together with utilizing explicit traceability links, facilitates bridging the gap between requirements and testing. Sabetta et al.discuss [35] that sometimes it might be needed to transform UML models into different analysis models which could each be used to verify (in a formal way) one kind of NFR. Some of these models are Petri nets, queuing networks, formal logic, etc. For this purpose, their abstraction-raising approach can transform UML models to different kind of analysis models in different formalisms. Hassan et al. focus on security requirements [36]. They propose the first goal-oriented software security engineering approach, Formal Analysis and Design for Engineering Security (FADES), aiming to produce software with high level of security in a systematic manner. FADES' support of automatic derivation of B formal method specifications and a suite of acceptance test cases from the requirements model ensures better alignment of security requirements and testing. Hussain and Eschbach present a model-based safety analysis approach [37] that automatically composes formal models of the system and produces a fault tree which can be used to generate test cases for the software system. Therefore test cases can be directly bound to the safety requirements and assure traceability between testing activity and safety requirements.

3.1.5. Code-centric approaches

Mugridge presents Storytest-Driven Development as a complementary form of TDD which can be applied to overall system development [38]. Storytests which are executable and business-oriented examples for each scheduled story are written by customers as an alternative to detailed requirements documents. As executable documents, they significantly reduce the need to derive independent tests because they help developers to continuously verify their consistency with the system. Baillon and Mongardé describe Behavior Driven Development (BDD) as a new development paradigm [39] in order to address traceability problems. They introduce the XReq tool which supports BDD in the Ada and other statically typed languages.

3.1.6. Problems and set of good practices in aligning requirements and testing

Uusitalo et al. present a set of good practices which can be applied to create a stronger link between requirements engineering and testing [3]. Some of these practices are involving testers during project planning and requirements reviews, which would lead to higher quality requirements and improved testability. A systematic approach is presented by Kukkanen et al. [1] for improving requirements and testing processes together with the aim of linking requirement and testing. They describe lessons learnt and best practices determined from applying new processes in an industrial case study. Sabaliauskaite et al. have carried out a survey in a large software company in Sweden, investigating the experienced obstacles in the alignment of requirements and testing [40].

3.1.7. Test cases

Nebut et al. concentrate on a guideline for automatic test case generation on embedded systems that are based on object oriented concepts [41]. The system requirements are described via use cases, contracts, and scenarios. If any other information for the requirements is needed, it is provided by different UML artifacts like sequence diagrams. Whalen et al. mention several problems of measuring the adequacy of black box testing using executable artifacts [42]. They also present coverage metrics based on formal high level software requirements. Conrad et al. presented a test case generation strategy which has been in use in an automotive company [43]. Siegl et al. are also interested in automotive industry proposed EXtended Automation Method (EXAM) for automatic generation of test cases, and the Timed Usage Model process for derivation of test cases from requirements [44]. Riebisch and Hubner concentrate on the first step of test case generation [45]. In this step their proposed method uses a description of the natural language and transforms it to an expression with formally defined syntax and semantics.

3.1.8. Approaches which support NFRs

Arnold et al. validation framework supports the modeling and automated validation of FRs and NFRs against several candidates IUTs [17]. In another research they have worked on a MBT approach which is based on Requirements Notation (URN). URN is one of few approaches that address the modeling and validation of both FRs and NFRs [46]. The approach proposed by Felderer et al. is suitable for testing Service level agreements (SLA) which is considered as

non-functional properties [6]. In their case study performance and security are included in modeling requirements. Duy et al. proposed framework – TROPOS is goal oriented in which NFRs are specified by softgoals [25]. For discovering aspects from goal models in AOSD, goals should be elicited and categorized to hard (FRs) and soft (NFRs) goals, hence AOSD can support NFRs [24]. In the method proposed by Méndez et al. specifying test cases (TCs) based on use cases (UCs) enables traceability between tests and FRs and NFRs [27]. In this approach the TCS Table can be used to define TC procedures associated to NFRs. The meta-model presented by Dubois et al. allows a full traceability of both FRs and NFRs through software development process [32]. In their research, Sabetta et al. transform UML models to different kind of analysis models, such as Petri nets, queuing networks, formal logic, etc. Each of these models could be used in formal verification of different NFRs [35]. Hassan et al. focus on alignment of security requirements and testing through supporting of automatic derivation of B formal method specifications and a suite of acceptance test cases from the requirements model [36]. In another research, Mugridge states that both Storytest-driven development and TDD depend on advanced automated testing techniques, including tests for non-functional requirements [38].

3.2. RQ2. What types of solutions are represented in these studies?

Figure 2 shows a map of existing research focusing on the alignment of requirements specification and testing, distributed over type of contribution and research type. It should be noted that a publication might provide multiple contributions e.g. both a tool and method.

3.3. RQ3. In which fora is research on alignment of requirement and testing published?

Distribution of research shows that most research is published in conferences and workshops 29/35(82%). There is also one book chapter and five journal publications (a full table of the distribution of publication fora can be found on the url http://www.cse.chalmers.se/~feldt/publications/alizadeh_2011_revvert.html). Distribution of publication types over time is shown in Table 3.

Figure 2. Map of research focus

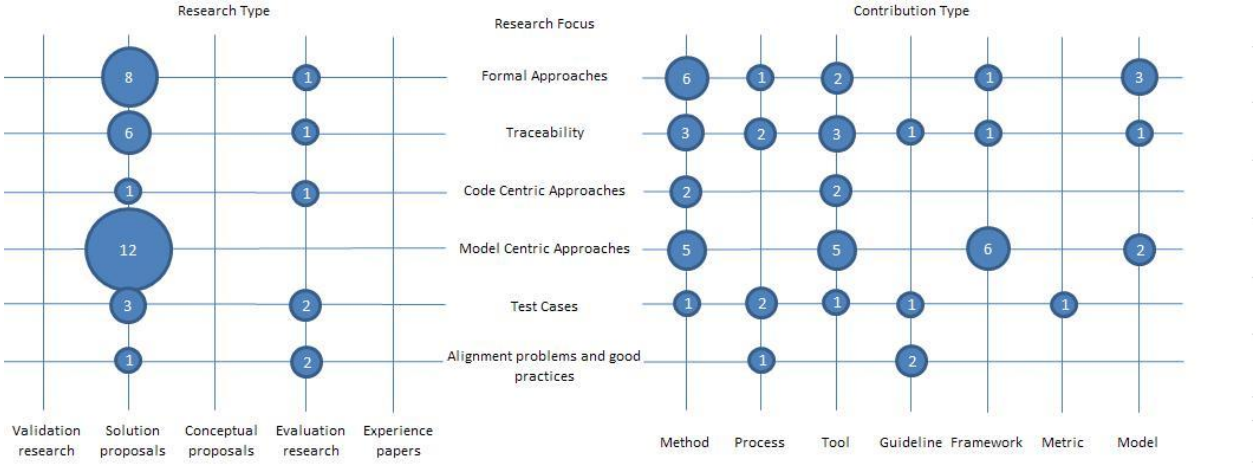

Table 3. Distribution of publication types over years

Publication Type	2002	2003	2004	2005	2006	2007	2008	2009	2010	Total
Conference	1	0	0	2	5	0	4	4	7	23
Workshop	0	0	0	3	1	1	1	0	0	6
Journal	0	1	0	0	1	0	1	0	2	5
Book	0	0	0	0	0	0	0	1	0	1
Total	1	1	0	5	7	1	6	5	9	35

4. Discussion

There are several challenges in using MBT approach for aligning requirements and testing. Researchers have tried to address these challenges. One challenge is to make test cases executable, as the tests are not at the same level of detail as the implementation code [17, 46]. Another challenge is to find interesting test cases. Test cases are said to be interesting if they covers requirements and can discover potential errors with a high probability [20]. Requirements traceability in MBT process is another important issue, which is the focus of several researches (such as [6, 20, 30]). In these researches the focus is mainly on FRs, and NFR traceability is still open for further research [26]. There are some researches on using MBT in service-oriented systems (like [6, 19]). The use of scenario notation for specification of system models has also attracted some attention (such as [17, 19, 30, 46]).

The focus on traceability issues is also conceivable since requirements traceability helps to determine the degree of test case coverage of requirements, and improves change management, which is crucially important to industry. As Abbors et al. mention [26] traceability reduces the time needed for debugging of the specification or the implementation of the system, by giving fast feedbacks.

Formal approaches which are another main research interest address translating informal requirements into formal models, generating tests from these formal models, and tracing between the informal requirements and tests [30]. Applying formal methods for aligning requirements and testing has some advantages and drawbacks. In this approach requirements are formulated in a precise, provable and correct representation. The representation is unambiguous and consistent [36]. This makes formal methods one of the best options for modeling and testing of safety critical systems. On the other hand using formal methods is difficult for practitioners [36]. Experienced people in this field are hard to come by and expensive to employ. The application of formal methods especially for large and complex systems is challenging because of their high cost and limited scalability. There is room for researches that combine the advantages of formal methods – formulating requirements in a precise, reliable and provable representation and the strength of informal methods – easy to learn and apply, to align requirement and testing.

Looking at figure 2 the research type in all research focus areas is solution proposal. This means that challenges in each research focus area are well understood, but the proposed solutions are just proposals and very little researches focus on the actual use and evaluation of proposals. Table 4 shows that only half of the papers have evaluated their ideas in industrial case studies, and their validity discussion is mostly medium (that is the author has mentioned threats to validity without a detailed discussion). In addition most of the searches are not mature enough to be published in journals. As mentioned in section 3.3, 29/35(82%) of researches are published in conferences and workshops and only 5 out of 35 papers are mature enough for a journal. All in all aligning the requirements and testing seems to be a rather immature area and is in need of more practical work.

The contribution type is mostly method 17/55(31%), tool 13/55(24%), and followed by framework. Presenting new methodologies or enhancing existing ones are needed to establish a strong link between requirements and testing. In order for them to be practical in industry, supportive tools and frameworks should be built.

Table 3 shows that interest in this field has grown in recent years, which could also serve as a motivation for more research.

Another important point is that most efforts in aligning requirements and testing have been on functional requirements. Table 5 shows that just 10/41(24%) of papers present approaches which could also be used for NFRs. This is a low percentage considering how important NFRs are in today's software systems.

Table 4. Validity discussion and case study

	#	%
Weak	7	20
Medium	17	49
Strong	11	31
Total	35	
With case study	16	46

Table 5. Distribution over FRs and NFRs

	#	%
FR	31	76
NFR	10	24
Total	41	

5. Conclusions

This paper presents a systematic mapping on aligning the specification and testing of functional or non-functional requirements. We identified 35papers published between 2001 and 2010, which address the following research questions: RQ1 - Which studies

have been done on linking the specification and testing of requirements? RQ2 - What types of solutions are represented in these researches? RQ3 - In which fora is research on alignment of requirement and testing published? To answer RQ1, the prior work on aligning requirements and testing is distributed in these focus areas: Model-centric approaches, Code-centric approaches, Traceability, Formal approaches, Test cases, Problems and set of good practices in aligning requirements and testing. The major focus is on MBT and traceability issues. In response to RQ2, The contribution type is mostly of "Method 17/55(26%)", "Tool 13/55(24%)", and followed by "Framework". In response to RQ3, most of the prior research has been published in conferences and workshops 29/35(82%). There is also one book chapter and five journal publications. Although industry is becoming more interested in establishing a strong link between requirements and testing, there is still a significant gap between these areas. This shows high potential for future work in establishing methods and processes with supportive tools. The current approaches in alignment have paid little attention to NFRs while they play a critical role in achieving successful software systems. As such, this area has high potential for further research.

References

[1] J. Kukkanen, K. Vakevainen, M. Kauppinen, *et al.*, "Applying a systematic approach to link requirements and testing: a case study," in *Asia-Pacific Software Engineering Conference (APSEC)*, Piscataway, NJ, USA, 2009, pp. 482-488.

[2] H. Post, C. Sinz, F. Merz, *et al.*, "Linking functional requirements and software verification," in *17thIEEE International Requirements Engineering Conference(RE)*, Piscataway, NJ, USA, 2009, pp. 295-302.

[3] E. J. Uusitalo, M. Komssi, M. Kauppinen, *et al.*, "Linking requirements and testing in practice," in *16th IEEE International Requirements Engineering Conference*, NJ, USA, 2008, pp. 265-70.

[4] Lars Grunske, "Specification Patterns for Probabilistic Quality Properties," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany.*, 2008, pp. 31-40.

[5] Abderrahman Matoussi and Régine Laleau, "A Survey of Non-Functional Requirements in Software Development Process," *Technical report TR-LACL-2008-7, LACL (Laboratory of Algorithms, Complexity and Logic), University of Paris-Est (Paris 12),* 2008.

[6] M. Felderer, P. Zech, F. Fiedler, *et al.*, "A Tool-based Methodology for System Testing of Service-oriented Systems," in *Second International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, Los Alamitos, CA, USA, 2010, pp. 108-13.

[7] K. Petersen, R. Feldt, S. Mujtaba, *et al.*, "Systematic mapping studies in software engineering," *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) University of Bari, Italy, 26–27 June,* 2008.

[8] B.A. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," *Technical Report EBSE-2007-01,* 2007.

[9] Lawrence Chung and Julio Cesar Prado Leite, "On Non-Functional Requirements in Software Engineering," in *Conceptual Modeling: Foundations and Applications*, T. B. Alexander, K. C. Vinay, G. Paolo, *et al.*, Eds., ed: Springer-Verlag, 2009, pp. 363-379.

[10] Marie-Agnes, Peraldi-Frati, and Arnaud Albinet, "Requirement traceability in safety critical systems," presented at the Proceedings of the 1st Workshop on Critical Automotive applications: Robustness and Safety, Valencia, Spain, 2010.

[11] J. Zou and C. J. Pavlovski, "Control cases during the software development life-cycle," in *IEEE Congress on Services Part 1 (SERVICES-1)*, Piscataway, NJ, USA, 2008, pp. 337-344.

[12] J. Metsa, M. Katara, and T. Mikkonen, "Testing Non-Functional Requirements with Aspects: An Industrial Case Study," in *Quality Software, 2007. QSIC '07. Seventh International Conference on*, 2007, pp. 5-14.

[13] Breno Lisi Romano, Glaucia Braga e Silva, Henrique Fernandes de Campos, *et al.*, "Software Testing for Web-Applications Non-Functional Requirements," presented at the Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, 2009.

[14] M. Glinz, "On Non-Functional Requirements," in *15th IEEE International Requirements Engineering Conference. RE '07.* , 2007, pp. 21-26.

[15] Roel Wieringa, Neil Maiden, Nancy Mead, *et al.*, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requir. Eng.,* vol. 11, pp. 102-107, 2005.

[16] M. Ivarsson and T. Gorschek, "Technology transfer decision support in requirements engineering research: a systematic review of REj," *Requirements Engineering, vol. 14, 2009, pp. 155-175.,* 2009.

[17] D. Arnold, J. P. Corriveau, and Shi Wei, "Modeling and validating requirements using executable contracts and scenarios," in *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*, CA, USA, 2010, pp. 311-20.

[18] A. Goel, B. Sengupta, and A. Roychoudhury, "Footprinter: Round-trip engineering via scenario and state based models," in *31st International Conference on Software Engineering - Companion Volume - ICSE-Companion*, Piscataway, NJ, USA, 2009, pp. 419-420.

[19] W. T. Tsai, R. Paul, L. Yu, *et al.*, "Scenario-Based Web Services Testing with Distributed Agents," *IEICE Transactions on Information and Systems,* vol. E86-D, pp. 2130-2144, 2003.

[20] C. Pfaller, A. Fleischmann, J. Hartmann, *et al.*, "On the integration of design and test: A model-based approach for embedded systems," in *Proceedings of the 2006 international workshop on Automation of software test(AST)* 2006, pp. 15-21.

[21] J. L. Boulanger and V. Q. Dao, "Requirements engineering in a model-based methodology for embedded automotive software," in *IEEE International Conference on Research, Innovation and Vision for the Future in Computing*

& Communication Technologies(RIVF), Ho Chi Minh City, Vietnam, 2008, pp. 263-268.

[22] L. O. Lobo and J. D. Arthur, "Local and global analysis: Complementary activities for increasing the effectiveness of requirements verification and validation," in *Proceedings of the 43rd Annual ACM Southeast Conference*, Kennesaw, GA, 2005, pp. 2256-2261.

[23] R. Marelly, D. Harel, and H. Kugler, "Multiple instances and symbolic variables in executable sequence charts," in *17th International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2002)*, USA, 2002, pp. 83-100.

[24] Niu Nan, Yu Yijun, B. Gonzalez-Baixauli*, et al.*, "Aspects across software life cycle: a goal-driven approach," in *Transactions on Aspect-Oriented Software Development. VI. Special Issue on Aspects and Model-Driven Engineering*, ed Berlin, Germany: Springer Verlag, 2009, pp. 83-110.

[25] Nguyen Duy Cu, A. Perini, and P. Tonella, "A goal-oriented software testing methodology," in *Agent-Oriented Software Engineering VIII. 8th International Workshop, AOSE 2007*, Berlin, Germany, 2008, pp. 58-72.

[26] F. Abbors, D. Truscan, and J. Lilius, "Tracing requirements in a model-based testing approach," in *2009 First International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, Piscataway, NJ, USA, 2009, pp. 123-8.

[27] E. Mendez, M. Perez, and L. E. Mendoza, "Improving software test strategy with a method to specify test cases (MSTC)," in *10th International Conference on Enterprise Information Systems. Databases and Information Systems Integration*, Setubal, Portugal, 2008, pp. 159-64.

[28] C. Quinn, S. Vilkomir, D. Parnas*, et al.*, "Specification of software component requirements using the trace function method," in *International Conference on Software Engineering Advances*, Tahiti 2006, pp. 50 - 50

[29] L. Naslavsky, T. A. Alspaugh, D. J. Richardson*, et al.*, "Using scenarios to support traceability," in *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering(TEFSE)* California,USA, 2005, pp. 25-30.

[30] A. Goel and A. Roychoudhury, "Synthesis and traceability of scenario-based executable models," in *2006 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006), 15-19 Nov. 2006*, Piscataway, NJ, USA, 2008, pp. 347-54.

[31] S. Ibrahim, M. Munro, A. Deraman*, et al.*, "A software traceability validation for change impact analysis of object oriented software," in *Proceedings of the International Conference on Software Engineering Research and Practice and Conference on Programming Languages and Compilers SERP'06*, USA, 2006, pp. 453-9.

[32] Hubert Dubois, Marie-Agnès Peraldi-Frati, and Fadoi Lakhal, "A model for requirements traceability in an heterogeneous model-based design process. Application to automotive embedded systems," in *15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Oxford, UK, 2010, pp. 233-242.

[33] F. Bouquet, C. Grandpierre, B. Legeard*, et al.*, "A subset of precise UML for model-based testing," in *Proceedings of the 3rd international workshop on Advances in modelbased testing (AMOST)*, 2007, pp. 95-104.

[34] J. Kelleher and M. Simonsson, "Utilizing use case classes for requirement and traceability modeling," in *Proceedings of the 17th IASTED International Conference on Modelling and Simulation*, Anaheim, CA, USA, 2006, pp. 617-25.

[35] A. Sabetta, D. C. Petriu, V. Grassi*, et al.*, "Abstraction-raising transformation for generating analysis models," in *Satellite Events at the MoDELS 2005 Conference. MoDELS 2005 International Workshops.* , Berlin, Germany, 2005, pp. 217-26.

[36] R. Hassan, M. Eltoweissy, S. Bohner*, et al.*, "Formal analysis and design for engineering security automated derivation of formal software security specifications from goal-oriented security requirements," *IET Software,* vol. 4, pp. 149-60, 2010.

[37] Tanvir Hussain and Robert Eschbach, "Automated Fault Tree Generation and Risk-Based Testing of Networked Automation Systems," in *Proceedings of 15th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 10)* Bilbao, Spain, 2010.

[38] R. Mugridge, "Managing agile project requirements with storytest-driven development," *IEEE Software,* vol. 25, pp. 68-75, 2008.

[39] C. Bâillon and S. Bouchez-Mongardé, "Executable requirements in a safety-critical context with Ada," *Ada User Journal,* vol. 31, pp. 131-135, 2010.

[40] G. Sabaliauskaite, A. Loconsole, E. Engstrom*, et al.*, "Challenges in Aligning Requirements Engineering and Verification in a Large-Scale Industrial Context," in *Requirements Engineering: Foundation for Software Quality. 16th International Working Conference, REFSQ*, Berlin, Germany, 2010, pp. 128-42.

[41] C. Nebut, F. Fleurey, Y. Le Traon*, et al.*, "Automatic test generation: a use case driven approach," *IEEE Transactions on Software Engineering,* vol. 32, pp. 140-55, 2006.

[42] M. W. Whalen, M. P. E. Heimdahl, A. Rajan*, et al.*, "Coverage metrics for requirements-based testing," in *Proceedings of the international symposium on Software testing and analysis(ISSTA)* 2006, pp. 25-35.

[43] M. Conrad, I. Fey, and S. Sadeghipour, "Systematic Model-Based Testing of Embedded Automotive Software," *Proceedings of the Workshop on Model Based Testing(MBT), Electronic Notes in Theoretical Computer Science,* vol. 111, pp. 13-26, 2005.

[44] Sebastian Siegl, Kai-Steffen Hielscher, and Reinhard German, "Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models," in *18th IEEE International Requirements Engineering Conference*, Sydney, New South Wales Australia, 2010.

[45] M. Riebisch and M. Hubner, "Traceability-driven model refinement for test case generation," in *Proceedings. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, CA, USA, 2005, pp. 113-20.

[46] D. Arnold, J. P. Corriveau, and Shi Wei, "Scenario-Based Validation: Beyond the User Requirements Notation," in *Software Engineering Conference (ASWEC), 2010 21st Australian*, 2010, pp. 75-84.

Extending Behavior Driven Development for Automated Testing of Probabilistic Non-Functional Requirements

Amir Hossein Ebrahimi
Department of Computer Science and Engineering
Chalmers University
Goteborg, Sweden
amirho@student.chalmers.se

Zeinab Alizadeh
Department of Computer Science and Engineering
Chalmers University
Goteborg, Sweden
barmi@student.chalmers.se

Robert Feldt
Department of Computer Science and Engineering
Chalmers University
Goteborg, Sweden
robert.feldt@chalmers.se

Abstract— ***There is increased emphasis on techniques to help test that software systems fulfill not only their functional but their non-functional requirements (NFR). Ideally such testing should be as closely based on a user specification as possible. Behavior-driven development (BDD) promotes such close alignment by basing system tests on behavioral scripts expressed closer to the terms and concepts of the users and the problem domain. We propose an extension, called Probabilistic BDD (ProBDD), to BDD framework in which probabilistic non-functional requirements can be expressed and then automatically tested. The extension is based on the ProProST NFR specification patterns which together with random generation of input data allows automated testing of NFR fulfillment. We describe the design and implementation of the ProBDD framework as well as an evaluation on two cases. Results show that the system utilizes the expressiveness of BDD and keeps a high level of traceability between NFR specification and testing.***

Keywords: Alignment; Behavior Driven Development; NFR; ProProST; Testing; Specification

I. INTRODUCTION

Requirement engineering (RE) and testing are essential activities that can directly determine the success or failure of software projects. Inadequate requirement specification and management such as incomplete or imprecise specification has led to failure in many software projects [1-2]. On the other hand, efficient testing activities are necessary in software projects since not only can they discover software faults, but they are also critical when evaluating reliability and performance [3]. Testing is a difficult and time consuming activity due to the increasing size and complexity of software systems [4]. However, if consistent testing is neglected during software development life cycle, the cost of maintenance would drastically increase [3]. A challenging issue in software development process is how to align requirement specification and testing. RE has a synergistic relationship with testing since requirements specify expectations on a software system and testing ensure these expectations are met [5]. Thus, to enable high product quality and efficient development requirements and testing activities need to be aligned. Although organizations are becoming more interested in establishing such a link, a gap often exists between requirements and testing.

With the ever-growing market demand for high quality software particularly in safety critical systems, the need and importance of efficient specification and testing of NFRs has become more apparent in recent years. Nevertheless, in the efforts on requirements specification, testing, and their alignment the focus is still mainly on functional requirements (FR)s while little attention is given to NFRs. NFRs are typically specified briefly and ambiguously and are left to be refined during architecture and design phase [6]. Often, NFRs are verified very late in the project, either after finishing the implementation [7] or during integration in traditional component based systems [8]. Also, the efforts to link requirements and testing lack a proper treatment of quality requirements. This results in many cases in lower quality of software. An example of consequences for such neglect is the famous case of the London Ambulance System in which NFRs non-compliance resulted in the deactivation of the system immediately after its deployment [9]. Among solutions which address such alignment are model based and formal approaches. There have also been other approaches in which alignment is supported via traceability between requirements and testing. Test Driven Development (TDD) as one of the approaches in this area has recently gained attention in industry. However, the idea of unit testing in TDD leads to structural division of the code instead of the a behavioral division [10]. This problem is addressed in BDD, as an evolution of TDD, in which the focus is on the behavioral aspects of software [10].

Our research question in this paper is “how and to what extent BDD scripts can be used to align specification and testing of NFRs?” The question is three-fold: 1. To what extent is it possible to specify NFRs with BDD scripts? 2. Which kind of NFRs can be modeled in this way? 3. Is it possible to test NFR fulfillment using these scripts? In order to address these questions we propose ProBDD, an extension to BDD, in which probabilistic NFRs - such as

safety, security, reliability, availability, and performance are specified in BDD scripts using NFR specification patterns from the ProProST framework [11]. ProProST is a repository of specification patterns for probabilistic properties with a structured English grammar as a textual front end [11]. As expressions made by ProProST do not sound natural for daily use, a few alternative patterns are added to ProBDD in order to make it more applicable. The proposed extended BDD scripts can be parsed and executed with randomly generated input data and test results can verify the specified NFRs fulfillment. Since ProProST patterns are specific to NFRs that can be written as probabilistic statements [11], our solution is specific to NFRs that are quantified and can be expressed in a probabilistic form. However, it seems unlikely that unquantified NFRs can ever be automatically tested without human interaction and the probabilistic framework is general enough to support a large variety of NFRs.

ProProST patterns help to capture expert knowledge and allow uniform specification of NFRs by different practitioners. The structured English grammar in ProProST helps non expert practitioners to specify probabilistic quality requirements correctly based on ProProST patterns. Using BDD scripts along with alternative patterns for ProProST increases NFRs specification expressiveness while minimizing miscommunication among different stakeholders. Automated testing of NFRs fulfillment establishes a high level of traceability between NFRs specification and testing.

The paper is organized as follows. Section 2 presents the background and related work to the alignment of NFRs specification and testing. Section 3 and 4 outline the design and implementation of the proposed framework. The framework is evaluated using two cases in Section 5. Discussion and conclusions are provided in sections 6 and 7, respectively.

II. BACKGROUND AND RELATED WORK

A. Related Work

In our systematic mapping paper [5], we investigated the existing research on the alignment of requirements specification and testing, published between 2001 and 2010. Some of the results consider NFRs in their proposed solutions. Among solutions on providing alignment, a great deal of attention has been paid to Model Based Testing (MBT). In MBT, models that generally facilitate communication between developers and stakeholders will contribute in software system testing as well [12]. For instance, Arnold et al. propose a validation framework which supports modeling and automated validation of both FRs and NFRs against several candidate Implementations Under Tests (IUTs) [13]. They claim that their approach supports the traceability between generated and executed test cases, and executions of an IUT for both FRs and NFRs. In another research they propose a scenario-driven MBT approach based on User Requirements Notation in which both FRs and NFRs could be modeled and validated [14]. As another instance, Felderer et al. propose a tool-based methodology for model-driven testing of service-oriented systems in a test–driven manner [12]. Their Telling TestStories tool which guarantees traceability between all types of modeling and system artifacts is suitable for testing Service Level Agreements (SLA) as a non-functional property. Formal approaches are another main research topic in this area [5]. They address translating informal requirements into formal models, generating tests from these formal models, and tracing between informal requirements and tests. Some formal approaches support NFRs. For instance, Sabetta et al. propose an approach to transform UML models into different analysis models which could each be used to verify (in a formal way) one kind of NFR [15]. Hassan et al. propose to align security requirements and testing through their formal framework, Formal Analysis and Design for Engineering Security (FADES) [16].

Both formal and model based approaches have some draw backs. Translating informal requirements into formal logic is challenging for non experts and leads to a decrease in communication between users and developing team. Moreover traceability management is difficult both in formal and model based approaches due to the gap between tests and requirements.

TDD supports alignment by forcing developers to always think about a small piece of functionality and write tests before developing the code that should be tested [17]. Since the main goal is to pass the test by writing the simplest solution, TDD forces simple bottom-up code development [17]. While writing tests, developers will be given a rapid feedback which leads to a high quality software project [17]. Storytest-Driven Development as a complementary form of TDD can be applied to overall system development [18]. Storytests are written by customers as an alternative to detailed requirements documents. They are executable and business-oriented examples for each scheduled story. As executable documents, they significantly reduce the need to derive independent tests because they help developers to continuously verify their consistency with the system.

B. Background

BDD is a requirement specification technique which is recently gaining more interest as a second generation agile methodology [19]. The three core principles in BDD are “Business and Technology should refer to the same system in the same way”, “Any system should have an identified, verifiable value to the business”, and “Up-front analysis, design and planning all have a diminishing return” [20]. TDD has some major drawbacks which result in an evolution into BDD. TDD’s aim is to define the behavioral intent of system by writing tests before

code [20]. However the idea of "unit" in this method leads to structural division of the code instead of the behavioral division[1] . This idea also involves developers in a detailed level of software programming such as testing the internal structure of the object being tested [10]. Hence, instead of thinking about the behavior of object and its interactions with other software components (what it *does*) they think about testing object structure (what an object is) [10]. In order to overcome this problem and help developers think about the solution in a more abstract level, BDD uses the terminology focused on behavioral aspects of software [20]. Its specific small vocabulary facilitates communication among stakeholders and assists them to consider the software system from the same perspective by using the same words [20]. Requirements in BDD are specified as features which should have business value for the customer using the following template [20]:

To gain a benefit,
As a Role,
I request a Feature

Alignment of requirements specification and testing in BDD is provided through connecting textual description of requirements to test [19]. Baillon and Mongardé describe BDD as a new development paradigm in order to address traceability problems [21]. They introduce XReq tool which supports BDD in Ada and other statically typed languages. Chelimsky et al. introduce RSpec as a Ruby-based BDD tool focused on specifying the behavior of system by writing executable examples of the expected behavior of a small bit of code in a controlled context [10]. In this way, Rspec can be used to specify requirements of software systems. However, Cucumber as another BDD tool moves one step forward in improving collaboration and communication between technical and business participants [10]. It is used to describe the application behavior at a higher level of abstraction without revealing its implementation details. The language of Cucumber is called Gherkin, a business-readable Domain Specific Language (DSL), in which application behavior is specified in "Given, When, Then" (GWT) format within feature files [22]. Feature files which are plain text files contain features with example scenarios each describe an acceptance criteria. Scenario step definitions should be implemented in Ruby scripts. Cucumber is able to parse feature files and invoke the step definitions to automate interaction with the code being developed [10]. The advantage of Cucumber is that it is easy for stakeholders to read and write feature files as they are written in a structured natural language. Hence, unlike traditional test driven development, the executable specifications in this methodology is actually treated as valuable documentation for both business experts and development team. Another advantage is the support for multiple languages in feature files, which also makes them valuable software documentation [10].

Besides the advantages of BDD it has some shortcomings regarding NFR specification. FRs, which are expectations of the system to be met, could be easily written in GWT format and are often of a binary nature, i.e. fulfilled or not. However, an NFR is often an expectation to be met within acceptable limits rather than absolutely and it is not straightforward to find a suitable place to put these limits within the GWT format. Hence, a supplementary pattern is needed to be integrated within BDD scripts to specify NFRs. One such pattern is provided by Grunske [11]. He proposes ProProST, a repository of specification patterns for probabilistic properties, written in the probabilistic temporal logic CSL (Continuous Stochastic Logic) including the following properties: "*Transient State Probability*", "*Steady State Probability*", "*Probabilistic Until*", "*Probabilistic Precedence*", "*Probabilistic Invariance*", "*Probabilistic Existence*", "*Probabilistic Response*", and "*Probabilistic Constrained Response*". As it is difficult for practitioners to handle formal specification, Grunske also proposes a structured English grammar as a textual front end for his pattern system [11]. These patterns have been developed using academic examples and properties from the industry in the domain of avionics, defense and automotive systems [11].

III. THE FRAMEWORK DESIGN

The goal of our framework is to align specification and testing of NFRs while giving practitioners the ability to specify requirements with natural English expressions. In this framework we use BDD scripts written with Cucumber, integrated with the ProProST patterns system to specify probabilistic NFRs, and a specialized testing tool to verify NFRs fulfillment. Figure 1 shows the overall design schema of the framework.

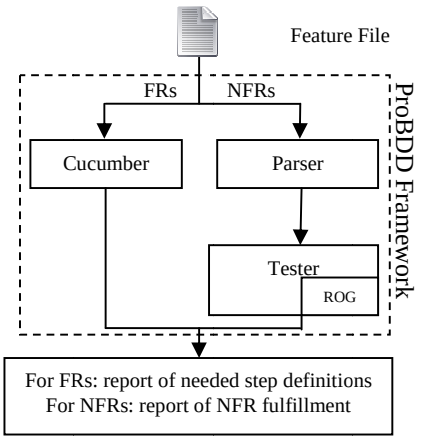

Figure 1. ProBDD Framework Design

[1] D. Astels, "A New Look at Test-Driven Development," in http://blog.daveastels.com/files/BDD_Intro.pdf, ed, Copyright © 2006.

A. *NFRs specification*

In order to specify NFRs in ProBDD, we integrate BDD scripts with ProProST specification pattern systems. The structured English grammar in ProProST helps non expert practitioners to specify probabilistic quality requirements correctly based on ProProST patterns. However, the requirements specified based on this grammar still have a degree of formality and do not sound natural; practitioners would not probably state requirements in this way. For instance, instead of writing: *"The system shall have a behavior were with a probability greater than 0.9 it is the case that 'a request is served' holds continuously within 10 time units"*, based on ProProST, it is easier and more readable to write: *"The probability that 'a request is served' within 10 time units is greater than 0.9"*. On the other hand, BDD is about writing requirements in a natural way. In order to make ProBDD more applicable for practitioners we add a few more alternative patterns which are more likely to be used in requirement specifications. These alternative patterns are just mapped to the ProProST patterns in the framework. From this point on the combined set of ProProST and alternative patterns is called ProBDD patterns. ProBDD patterns enable users to specify their requirements with short natural English expressions. In order to develop these patterns we partially used the Grunske survey on academic NFR examples and real world quality requirements from avionics, defense and automotive systems [11]. *"ProbabilisticInvariance"* ProProST pattern and its alternatives are given as an example in Table 1.

probabilisticInvariance	:==	'the system shall have a behavior where with a probability' probabilityBound 'it is the case that' stateFormula 'holds continuously' timeBound
alternativeOne	:==	'the' probability of stateFormula timeBound 'is' probabilityBound
alternativeTwo	:==	'the' probability 'is' probabilityBound 'that' stateFormula timeBound
alternativeThree	:==	'with' ['a'] probability ['of'] probabilityBound stateFormula timeBound
alternativeFour	:==	'there is' ['a'] probability ['of'] probabilityBound of stateFormula timeBound
alternativeFive	:==	'with' aProbabilityBound probability stateFormula timeBound
alternativeSix	:==	'there is' aProbabilityBound probability of stateFormula timeBound
alternativeSeven	:==	stateFormula timeBound ['in'] probabilityBound ofTheTime
alternativeEight	:==	'in' probabilityBound ofTheTime stateFormula timeBound
alternativeNine	:==	stateFormula timeBound 'with' ['a'] probability ['of'] probabilityBound
Probability	:==	'probability' | 'chance'
probabilityBound	:==	(atBound | thanBound) $p \in R^{[0,1]}$
aProbabilityBound	:==	(atBound ['a'] | ['a'] thanBound) $p \in R^{[0,1]}$
atBound	:==	'at most' | 'at least'
thanBound	:==	lowerOrEqualThan | higherOrEqualThan | greaterThan | lowerThan | equalTo
lowerOrEqualThan	:==	('lower' | 'less') ('or equal than' | 'than or equal to')
higherOrEqualThan	:==	('greater' | 'higher') ('or equal than' | 'than or equal to')
greaterThan	:==	'greater than' | 'higher than'
lowerThan	:==	'lower than' | 'less than'
equalTo	:==	['equal to']
of	:==	'of' | 'to' | 'that' | 'in which'
timeBound	:==	upperTimeBound | lowerTimeBound | timeInterval | noTimeBound
upperTimeBound	:==	('within the next' | 'in less than' | 'in at most' | 'within' | 'in' | 'before') $t \in R^{>0}$ timeUnits
lowerTimeBound	:==	('after' | 'in more than' | 'in at least') $t \in R^{>0}$ timeUnits
timeInterval	:==	'between' $t \in R^{>0}$ 'and' $t \in R^{>0}$ timeUnits
noTimeBound	:==	''
timeUnits	:==	'time units' | 'time steps'
ofTheTime	:==	'of the' ('cases' | 'time')
stateFormula	:==	SimpleStateProperty

Table 1 - Alternative patterns for probabilisticInvariance

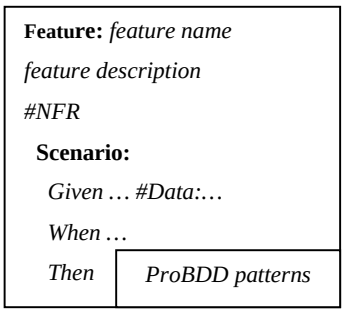

Figure 2. ProBDD specification schema

NFRs can be categorized in two groups: 1. NFRs which place constraints on a specific functionality of the system, 2. NFRs which are not concerned with a specific functionality and put constraints on the software system as a whole. Both groups can be specified by features and feature scenarios. In the first group, system functionalities (e.g. search functionality) can be specified by features. Feature scenarios are then used to describe the acceptance criteria (e.g. a scenario for successful search) and also constraints on the functionality (e.g. a scenario for search time limit). For the second group (e.g. system stability), features and feature scenarios are used to explain the global constraints on the software system.

Feature files should follow the ProBDD specification schema, given in Figure 2. As this Figure shows, feature scenarios have three parts: "*Given*", "*When*", and "*Then*". "*Given*" shows preconditions i.e. the state of environment before an action holds. The action is represented in "*When*" clause. Constraint on the action and the probability of its fulfillment is specified using one of the ProBDD patterns in "*Then*" clause. As the majority of probabilistic NFRs follow the "*probabilisticInvariance*", "*probabilisticResponse*" and "*probabilisticExistence*" ProProST patterns [11], these patterns are currently implemented in ProBDD. However, our framework is general enough to implement the other kinds of ProProST patterns.

An example of the first group is shown below in which "*Item Search*" is an FR, "*Successful item search*" is an acceptance criterion, and "*Search time limit*" is an NFR which is specified using the "*alternativeSeven"* pattern.

***Feature**: Item Search*

In order to find an item in a list

As a user

I want to search the list

***Scenario**: Successful item search*

Given any random list

And an item is inserted in the list

When I search for that item

Then I should receive its index

And I should see the message saying "Item found"

***Scenario**: Search time limit*

Given any random list

When I search a random item

Then "the search is completed" within 10 seconds 90% of the time

From the second group, "*System Availability*" describes availability NFR with "*Availability after shutdown*" as an example of NFR scenario, which is specified using "*alternativeThree*" pattern:

***Feature**: System Availability*

In order to have a high quality system

As a user

I want system to be available

***Scenario**: Availability after shutdown*

When a shutdown occurs

Then with a probability greater than 75% "system recovery is completed" between 100 and 200 seconds.

B. NFRs testing

An NFR scenario is considered testable by ProBDD if its specification is based on ProBDD patterns. Hence, the parser in the framework is responsible to verify each NFR scenario's conformance to one of the ProBDD patterns. Parsing of FRs scenarios is left to Cucumber. The parser is also responsible for extracting required parameters for testing from "*Then*" clause of testable NFRs. For instance in "*probabilisticInvariance*" pattern the parameters to be extracted are "*stateFormula*" (name of the action which is going to be tested), "*probabilityBound*" (the expected probability with which the action will hold in a time bound), "*probabilityOperator*", and "*timeBound*". These parameters are used by the tester part of the framework to verify NFR fulfillment. It should be noticed that since NFRs are specified in probabilistic manner using ProBDD patterns, they are also in need of probabilistic testing i.e. using randomly generated data. Generating such data is the responsibility of Random Object Generator (ROG) component. As random objects' structure could vary for each "*stateFormula*", ROG is responsible for generating random data based on the structure specified by user of the framework for each "*stateFormula*". The tester monitors the "*stateFormula*" results for a series of randomly generated test data several times, each time verifying if the "*stateFormula*" will hold in the "*timeBound*". Then it calculates the success probability and compares it with "*probabilityBound*" based on the "*probabilityOperator*" to verify NFR fulfillment.

IV. THE FRAMEWORK (IMPLEMENTATION)

An overall view of the ProBDD implementation is given in Figure 3. The implementation follows the same structure as any Cucumber project [10] which is shown in Figure 4. A set of "*features"* describe functional requirements of the framework. These features contain

scenarios specifying their acceptance criteria. The implementation of scenario steps is given in "*step definitions*". "*Spec*" files describe the behavior of framework components. These files contain executable examples of the expected behavior of components in a controlled context. "*Lib*" contains framework components.

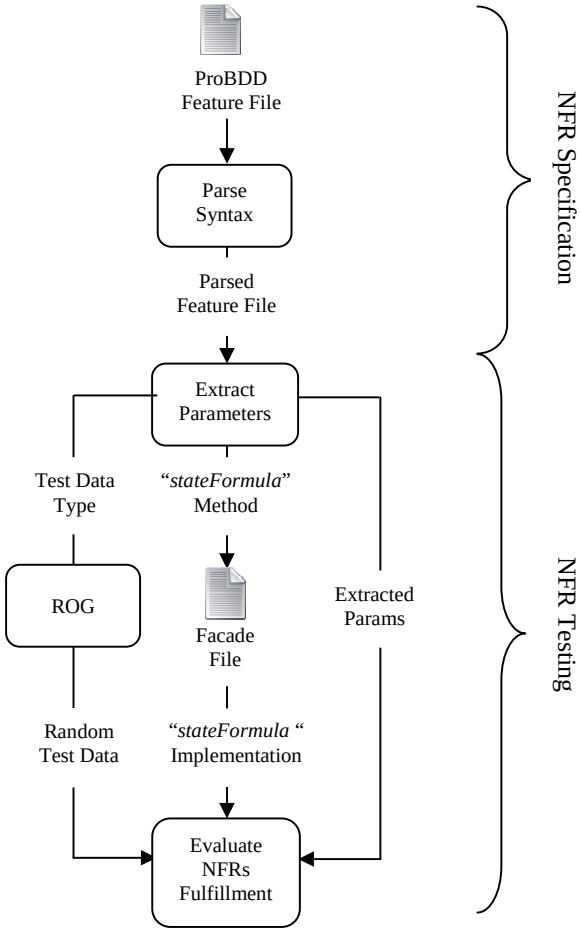

Figure 3. Overall view of ProBDD Framework Implementation

Framework's features and their descriptions are mentioned below:

Feature: NFRs specification are syntactically checked
Description: For a given feature file, NFR scenarios are recognized with "*#NFR*" tags before "*Scenario*" keyword. For each NFR scenario their "*Then*" clause is parsed to check it syntactically against ProBDD patterns. In order to implement these patterns we use the "*Treetop*" ruby library for two reasons[2] . Firstly, "*Treetop*" allows you to easily create parsers using a "*Parser Expression Grammar*" (PEG) [23]. Parser's behavior can be described by a set of rules and its structure is described using a set of "*Treetop*" classes. Secondly, as Gherkin (the language that Cucumber understands) is also implemented using "*Treetop*", ProBDD framework would be more compatible with cucumber.

Feature: Parameters and file needed for NFR verification is prepared
Description: In this feature parameters needed for NFR testing are extracted and also a façade file is created: a. parameters needed for NFR testing are categorized into three groups. First group includes the expected structure(s) of random input data which is specified by the user of the framework using an array of multiple types. This group is recognized with "*#data*" in the "Given" clause. For instance, if the user wants to test the "*stateFormula*" with random lists of type [integer, integer] or [integer, float] the corresponding script would be:

#data: [integer, integer] | [integer, float]

Second group are parameters extracted from "*Then*" clause like "*stateFormula*", "*probabilityBound*", "*timeBound*" etc. Third group are any extra parameters which are needed for the implementation of "*stateFormula*" by the user. They are recognized with "*optional_params*" keyword in the "*Given*" clause (after the structure of random inputs). The use of this group is optional. b. In order to test NFR fulfillment user should be able to give an implementation of "*stateFormula*" to the framework to execute. A Façade file is created in this feature to act as a communication vehicle through which user can specify this implementation. This file includes a method with "*randomData*" argument and an empty body per "*stateFormula*". So, users will be able to add the implementation of the "*stateFormula*" in its corresponding method.

Feature: NFR fulfillment is verified
Description: For each implemented "*stateFormula*", the framework performs monitoring several times. Monitoring period corresponds to the specified "*timeBound*". During each monitoring the corresponding method of "*stateFormula*" in the facade file is executed given a series of randomly generated data, provided by ROG. Then, the probability with which "*stateFormula*" meets the stated constraints within monitoring period is calculated. This probability is checked against the "*probabilityBound*" based on the "*probabilityOperator*". At the end a report is generated to state the results of these tests.

Random Object Generator (ROG)

ROG is responsible for generating random test data with a given structure. "*Rant*" library[3] is used to generate random objects as it has methods each returning a random

[2] *Tree top*. Available: http://treetop.rubyforge.org/

[3] *Rant Library*. Available: https://github.com/hayeah/rantly

value of some type (e.g. string, integer, float, etc.). The "*branch*" method of "*Rant*" library is used to generate random data with more than one expected type. According to the possible complexity of some test data structure, their generation might take some time. This time is not included in duration of the monitoring period.

V. EVALUATION

For a preliminary evaluation of ProBDD framework we looked for algorithms which are: of a probabilistic nature, are used practically, have applications in industry, and have list of objects as parameter. We came up with algorithms in artificial intelligence area, one of them is clustering. "Clustering techniques attempt to group points in a multidimensional space in such a way that all points in a single group have a natural relation to one another and points not in the same group are somehow different" [24]. These techniques are practically used in engineering, medical, biological, and many other areas. We chose well known "*Single Linkage*" clustering since it is recognized as a theoretical foundation of cluster analysis, gets a list of objects to be clustered, has a known complexity of $O(n^2)$ for an optimally efficient algorithm known as "*Slink*" [25], and has an implementation in ruby ("*Ai4r*" ruby library)[4].

Performance NFR is a good choice for evaluating the framework, as it is quite straightforward to specify and test performance quality requirement from the beginning of the development process. Additionally, performance can be specified by "*probabilisticExistence*" and "*probabilisticInvariance*" patterns which are both implemented in this framework [11]. The goal is to specify and test two performance NFR cases with relatively known validity for Single Linkage algorithm (one correct and one incorrect). ProBDD framework should return the same validity result for these NFRs.

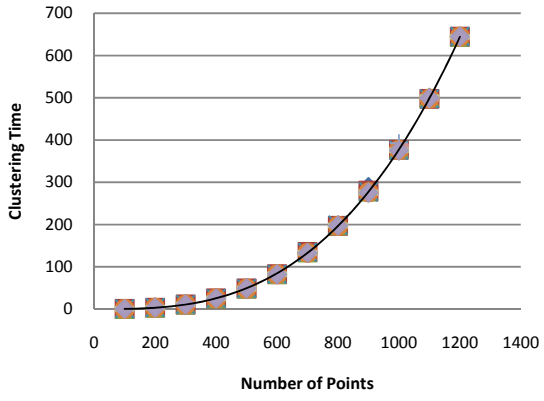

Figure 4. Single linkage algorithm performance

[4] *Ai4r Ruby library*. Available: http://ai4r.rubyforge.org/

To find these cases we model the behavior of the algorithm by running it ten times for different number of points to get an approximation of its running time. Figure 5 shows the clustering performance for different number of points. We pick two points (n = 700 and n = 800) and estimate their clustering time with the help of the graph. Based on these estimations we specify a feature for clustering points in which the NFR scenario named "*Performance of clustering 700 points*" is valid and another one "*Performance of clustering 800 points*" is invalid.

***Feature**: Clustering Points*

In order to group points in such a way that all points in a single group have a natural relation to one another

As a user

I want to cluster points

#NFR

***Scenario**: Performance of clustering 700 points*

Given any 700 points #data: array(700){[integer ,integer]}

When I want to cluster the points

Then "clustering of points is completed" within 134 seconds 60% of the times.

#NFR

***Scenario**: Performance of clustering 800 points*

Given any 800 points #data: array(800){[integer, integer]}

When I want to cluster the points

Then "clustering of points is completed" within 100 seconds 30% of the times.

This feature file is tested using the ProBDD framework. The Façade file includes implementation of method corresponding to *"clustering of points is completed"* as shown below. In this method "*Ai4r*", a ruby library which includes several artificial intelligence algorithms, is used to implement Single Linkage algorithm.

```
def clustering_of_points_is_completed(data,
optional_Params)
        result = false
        require 'Ai4r'
        ai4r_data =
        i4r::Data::DataSet.new(:data_items=>data)
        b = Ai4r::Clusterers::SingleLinkage.new()
        b.build(ai4r_data, 4)
        result = true
        return result
end
```

Testing results, shown in Figure 6 and summarized in Table 1 are promising. The results show that the first NFR fulfillment is verified by the framework, as the desired response time is near the actual behavior of the clustering algorithm. On the other hand, the second NFR fulfillment is not verified by the framework since the desired response time is considerably lower than the actual behavior of the algorithm. The calculated success probability, given in the results, can help the user to modify their specified "*probabilityBound*" or "*timeBound"* to fulfill the NFR. Additionally user can put different implementation of clustering in "*clustering_of_points_is_completed*", and compare their results to choose the best implementation which fits their needs. These findings confirm that ProBDD provides an interactive environment for users to specify and test their probabilistic NFRs.

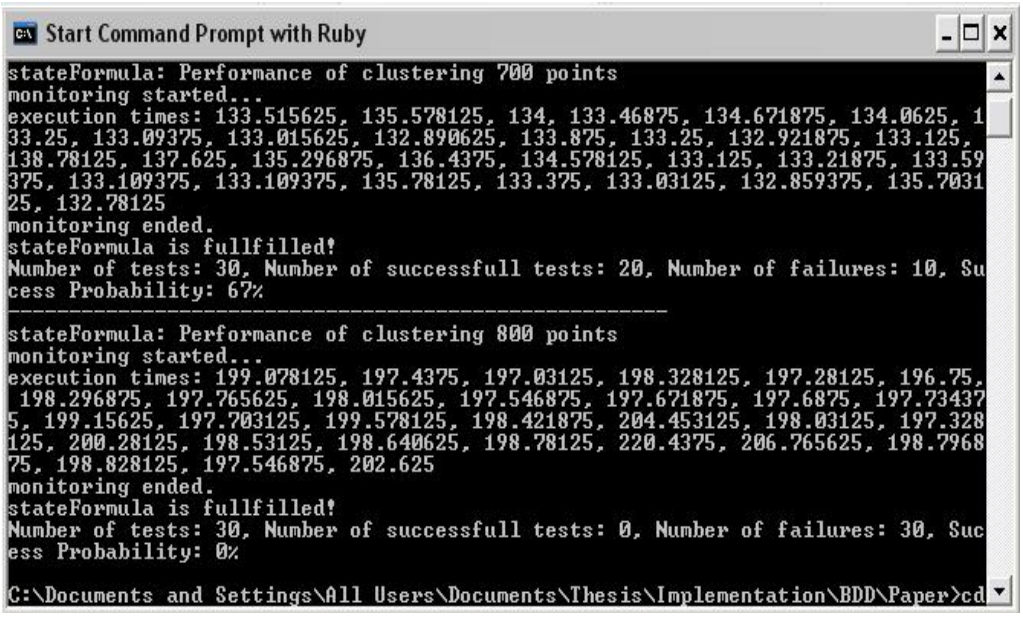

Figure 5. Cases verification results

Clustering Points	#Test	#Success	#Fail	Success Probability
700	30	20	10	67%
800	30	0	30	0%

Table 2. Evaluation results

VI. DISCUSSION

ProBDD is an extension to BDD framework, in which probabilistic NFRs are specified in BDD scripts using NFR specification patterns from the ProProST framework together with some alternative patterns. These scripts serve as automatic tests to verify their specified NFRs fulfillment with a series of randomly generated data. As the scripts are used for both specification and test of NFRs, their alignment is automatically reached while maintaining a high level of traceability. As ProProST patterns are specific to formulate probabilistic NFRs, ProBDD is specific to NFRs that are quantified and can be expressed in a probabilistic form. However, it seems unrealistic to automatically test unquantified NFRs without user interaction and the probabilistic framework is general enough to support a large variety of NFRs. ProBDD benefits from the advantages of BDD, ProProST patterns and their alternatives. BDD's small non technical vocabulary makes requirements simple to write and easy to read. These requirements reflect the behavioral intent of the software without focusing on implementation details. As a consequence the communication and collaboration will be increased between non technical users and the development team. Increasing the level of common understanding leads to saving in time and money while reducing the costs of future change. As the scripts are written in the same language of the problem domain they are a useful source of documentation for the resulting system. ProProST patterns are used within BDD scripts to provide a suitable solution for NFR specification. These patterns help non-expert practitioners to correctly

specify probabilistic NFRs and also allow a uniform specification of NFRs by different practitioners. The addition of some alternative patterns which are more likely to be found in daily use, as the front end to ProProST patterns, make ProBDD more applicable and easier to use. Current BDD approaches enable specification of executable requirements; however, they have not specifically addressed how to write executable NFRs. On the other hand, ProBDD enables users to express NFRs in a precise manner. Our BDD-based approach differs from formal approaches regarding the requirements specification simplicity. Formal approaches should translate informal requirements into formal models upon which tests could be generated. This translation is often challenging for non experts. On the other hand, in ProBDD NFRs are specified in structured natural language right from the beginning which simplifies the communication between users and development team. ProBDD provides a better support for traceability management than formal and MBT approaches. Traceability management is difficult in formal approaches due to the gap between requirements and test cases. A gap is also evident in MBT approaches, in which test cases are generated based on models instead of requirements. However, our approach bridges this gap between requirement specification and testing and keeps a perfect traceability between them. This is a result of stating NFR specification and their acceptance test in the same script.

We think that the proposed approach has the potential to specify and test a large variety of probabilistic NFRs. However, writing tests for these NFRs could sometimes be problematic (e.g. safety requirements in real time systems). ProProST pattern system is based on specific domains such as avionics, defense, and automotive systems. The application of these patterns to other domains is yet to be validated. The approach has been evaluated with two cases; however it should be evaluated in an industrial setting in order for it to be practically used.

VII. CONCLUSION

This research sets out to answer the question “how and to what extent BDD scripts can be used to align specification and testing of NFRs?” through presenting an extension to BDD framework, ProBDD, to specify executable probabilistic NFRs. Automated testing of NFRs fulfillment is supported through the integration of ProProST NFRs specification patterns and their alternatives into BDD scripts together with random generation of test data. The goal of the study is to align NFRs specification and testing and maintain a high level of traceability between them, while providing necessary tools for the specification of requirements with short natural English expressions. ProBDD framework has advantages each of which originate from one of its contributors. Using BDD scripts makes requirement specification simple, readable, and easy to communicate among different stakeholders. As these scripts are written in the same language of problem domain, they would make perfect sense to non technical users. So, unlike traditional TDD approaches, these specifications can really act as valuable system documentation. The use of ProProST patterns enables the capture of expert knowledge and also allows a uniform specification of probabilistic NFRs by different practitioners. The alternative patterns help to make ProBDD more applicable and useable. The proposed solution is evaluated through testing performance NFR in two cases with known validity. However, an obvious candidate for further work is to evaluate the framework in industrial setting. Other interesting directions to pursue are, a. evaluating the framework for other kinds of probabilistic NFRs b. investigating possible patterns for non-probabilistic NFRs to extend the framework with.

VIII. ACKNOWLEDGMENT

We wish to thank Ali Shahrokni for his useful comments and ideas, and his help with some of the figures and examples in the paper.

IX. REFERENCES

[1] L. Chirinos, *et al.*, "Characterizing a data model for software measurement," *Journal of Systems and Software - Special issue: The new context for software engineering education and training*, vol. 74 2005.

[2] M. Jaffe and N. Leveson, "Completeness, robustness, and safety in real-time software requirements specification," in *ICSE Proceedings of the 11th international conference on Software engineering*, 1989.

[3] W. Dulz and Z. Fenhua, "MaTeLo - statistical usage testing by annotated sequence diagrams, Markov chains and TTCN-3," in *Third International Conference on Quality Software, 2003. Proceedings.* , 2003, pp. 336-342.

[4] A. Saifan and J. Dingel, "A Survey of Using Model-Based Testing to Improve Quality Attributes in Distributed Systems," in *Advanced Techniques in Computing Sciences and Software Engineering*, K. Elleithy, Ed., ed: Springer Netherlands, 2010, pp. 283-288.

[5] Z. A. Barmi, *et al.*, "Alignment of requirements specification and testing: A systematic mapping study," in *Fourth International Conference on Software Testing, Verification and Validation Workshops*, Berlin,Germany, 2011.

[6] J. Zou and C. J. Pavlovski, "Control cases during the software development life-cycle," in *IEEE Congress on Services Part 1 (SERVICES-1)*, Piscataway, NJ, USA, 2008, pp. 337-344.

[7] A. Matoussi and R. Laleau, "A Survey of Non-Functional Requirements in Software Development

Process," *Technical report TR-LACL-2008-7, LACL (Laboratory of Algorithms, Complexity and Logic), University of Paris-Est (Paris 12)*, 2008.

[8] J. H. Hill, *et al.*, "Unit Testing Non-functional Concerns of Component-based Distributed Systems," presented at the Proceedings of the 2009 International Conference on Software Testing Verification and Validation, 2009.

[9] D. R. Lindstrom, "Five Ways to Destroy a Development Project," *IEEE Software*, vol. 10, pp. 55 - 58 1993.

[10] D. Chelimsky, *et al.*, *The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends*, 2009.

[11] L. Grunske, "Specification patterns for probabilistic quality properties," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008, pp. 31-40.

[12] M. Felderer, *et al.*, "A Tool-based Methodology for System Testing of Service-oriented Systems," in *Second International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, Los Alamitos, CA, USA, 2010, pp. 108-13.

[13] D. Arnold, *et al.*, "Modeling and validating requirements using executable contracts and scenarios," in *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*, CA, USA, 2010, pp. 311-20.

[14] D. Arnold, *et al.*, "Scenario-Based validation: Beyond the user requirements notation," in *Australian Software Engineering Conference (ASWEC)*, Los Alamitos, CA, USA, 2010, pp. 75-84.

[15] A. Sabetta, *et al.*, "Abstraction-raising transformation for generating analysis models," in *Satellite Events at the MoDELS 2005 Conference. MoDELS 2005 International Workshops.* , Berlin, Germany, 2005, pp. 217-26.

[16] R. Hassan, *et al.*, "Formal analysis and design for engineering security automated derivation of formal software security specifications from goal-oriented security requirements," *IET Software*, vol. 4, pp. 149-160, 2010.

[17] D. H. Steinberg and D. W. Palmer, *Extreme Software Engineering A Hands-On Approach* 2003.

[18] R. Mugridge, "Managing agile project requirements with storytest-driven development," *IEEE Software*, vol. 25, pp. 68-75, 2008.

[19] R. A. d. Carvalho, *et al.*, "Filling the Gap between Business Process Modeling and Behavior Driven Development."

[20] *Behaviour-Driven Development*. Available: http://behaviour-driven.org/

[21] C. Bâillon and S. Bouchez-Mongardé, "Executable requirements in a safety-critical context with Ada," *Ada User Journal*, vol. 31, pp. 131-135, 2010.

[22] M. Wynne and A. Hellesøy, *The Cucumber Book: Behaviour-Driven Development for Testers and Developers (Betta book)*.

[23] B. Ford, "Parsing expression grammars: a recognition-based syntactic foundation," presented at the Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Venice, Italy, 2004.

[24] R. Dubes and A. K. Jain, "Clustering techniques: The user's dilemma," *Pattern Recognition*, vol. 8, pp. 247-260, 1976.

[25] R. Sibson, "SLINK: an optimally efficient algorithm for the single-link cluster method," *The Computer Journal (British Computer Society)*, vol. 16 (1), pp. 30-34, 1973.

References:

[1] Andrew Stellman and Jennifer Greene, *Applied software project management*: O'Reilly Media, Inc., 2005.

[2] E. J. Uusitalo, M. Komssi, M. Kauppinen, *et al.*, "Linking requirements and testing in practice," in *16th IEEE International Requirements Engineering Conference*, NJ, USA, 2008, pp. 265-70.

[3] J. Kukkanen, K. Vakevainen, M. Kauppinen, *et al.*, "Applying a systematic approach to link requirements and testing: a case study," in *Asia-Pacific Software Engineering Conference (APSEC)*, Piscataway, NJ, USA, 2009, pp. 482-488.

[4] J. Zou and C. J. Pavlovski, "Control cases during the software development life-cycle," in *2008 IEEE Congress on Services Part 1 (SERVICES-1), 6-11 July 2008*, Piscataway, NJ, USA, 2008, pp. 337-44.

[5] Abderrahman Matoussi and R´egine Laleau, "A Survey of Non-Functional Requirements in Software Development Process," *Technical Report*, 2008.

[6] D.R. Lindstrom, "Five Ways to Destroy a Development Project," *IEEE Software*, vol. 10, pp. 55 - 58 1993.

[7] Rogerio Atem de Carvalho, Rodrigo Soares Manhaes, and Fernando Luis de Carvalho e Silva, "Filling the Gap between Business Process Modeling and Behavior Driven Development."

[8] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," *Technical Report EBSE-2007- 01*, 2007.

[9] K. Petersen, R. Feldt, S. Mujtaba, *et al.*, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) University of Bari, Italy, 26–27 June*, 2008.

[10] L. Grunske, "Specification patterns for probabilistic quality properties," in *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008, pp. 31-40.