Dietmar Winkler
Stefan Biffl
Johannes Bergsmann (Eds.)

Software Quality

Software and Systems Quality in Distributed and Mobile Environments

LNBIP 200

7th International Conference, SWQD 2015
Vienna, Austria, January 20–23, 2015
Proceedings

Springer

Lecture Notes in Business Information Processing 200

More information about this series at http://www.springer.com/series/7911

Dietmar Winkler · Stefan Biffl
Johannes Bergsmann (Eds.)

Software Quality

Software and Systems Quality in Distributed and Mobile Environments

7th International Conference, SWQD 2015
Vienna, Austria, January 20–23, 2015
Proceedings

Springer

Editors
Dietmar Winkler
Vienna University of Technology
Vienna
Austria

Johannes Bergsmann
Software Quality Lab GmbH
Linz
Austria

Stefan Biffl
Vienna University of Technology
Vienna
Austria

ISSN 1865-1348 ISSN 1865-1356 (electronic)
Lecture Notes In Business Information Processing
ISBN 978-3-319-13250-1 ISBN 978-3-319-13251-8 (eBook)
DOI 10.1007/978-3-319-13251-8

Library of Congress Control Number: 2014956222

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Message from the General Chair

The Software Quality Days (SWQD) conference and tools fair started in 2009 and has grown to one of the biggest conferences on software quality in Europe with a strong community. The program of the SWQD conference is designed to encompass a stimulating mixture of practical presentations and new research topics in scientific presentations as well as tutorials and an exhibition area for tool vendors and other organizations in the area of software quality.

This professional symposium and conference offer a range of comprehensive and valuable opportunities for advanced professional training, new ideas, and networking with a series of keynote speeches, professional lectures, exhibits, and tutorials.

The SWQD conference is suitable for anyone with an interest in software quality, such as test managers, software testers, software process and quality managers, product managers, project managers, software architects, software designers, user interface designers, software developers, IT managers, development managers, application managers, and similar roles.

January 2015 Johannes Bergsmann

Message from the Scientific Program Chair

The Seventh Software Quality Days (SWQD) conference and tools fair brings together researchers and practitioners from business, industry, and academia working on quality assurance and quality management for software engineering and information technology. The SWQD conference is one of the largest software quality conferences in Europe.

Over the past years a growing number of scientific contributions were submitted to the SWQD symposium. Starting in 2012 the SWQD symposium included a dedicated scientific track published in scientific proceedings. For the fourth year we received an overall number of 13 high-quality submissions from researchers across Europe, which were each peer-reviewed by three or more reviewers. Out of these submissions, the editors selected four contributions as full papers, an acceptance rate of 31 %. Further, four short papers, which represent promising research directions, were accepted to spark discussions between researchers and practitioners at the conference.

The main topics from academia and industry focused on systems and software quality management methods, improvements of software development methods and processes, latest trends in software quality, and testing and software quality assurance.

This book is structured according to the sessions of the scientific track following the guiding conference topic "Software and Systems Quality in Distributed and Mobile Environments":

- Risk Management and Inspection
- Change Impact Analysis and Systems Testing
- Software and Systems Architectures

January 2015 Stefan Biffl

Organization

SWQD 2015 has been organized by Software Quality Lab GmbH and the Vienna University of Technology, Institute of Software Technology and Interactive Systems, and the Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems."

Organizing Committee

General Chair

Johannes Bergsmann	Software Quality Lab GmbH, Austria

Scientific Program Chair

Stefan Biffl	Vienna University of Technology, Austria

Proceedings Chair

Dietmar Winkler	Vienna University of Technology, Austria

Organizing and Publicity Chair

Petra Bergsmann	Software Quality Lab GmbH, Austria

Program Committee

SWQD 2015 established an international committee of well-known experts in software quality and process improvement to peer-review the scientific submissions.

Maria Teresa Baldassarre	University of Bari, Italy
Miklos Biro	Software Competence Center Hagenberg, Austria
Matthias Book	University of Duisburg-Essen, Germany
Ruth Breu	University of Innsbruck, Austria
Fabio Calefato	University of Bari, Italy
Maya Daneva	University of Twente, The Netherlands
Oscar Dieste	Universidad Politécnica de Madrid, Spain
Frank Elberzhager	Fraunhofer IESE, Germany
Michael Felderer	University of Innsbruck, Austria
Gordon Fraser	University of Sheffield, UK
Marcela Genero	University of Castilla-La Mancha, Spain
Volker Gruhn	University of Duisburg-Essen, Germany
Jens Heidrich	Fraunhofer IESE, Germany
Frank Houdek	Daimler AG, Germany

Slinger Jansen	Utrecht University, The Netherlands
Marcos Kalinowski	Federal University of Juiz de Fora (UFJF), Brazil
Petri Kettunen	Helsinki University, Finland
Mahvish Khurum	Blekinge Institute of Technology, Sweden
Ricardo Machado	Universidade do Minho, Portugal
Eda Marchetti	ISTI-CNR, Italy
Paula Monteiro	Universidade do Minho, Portugal
Juergen Muench	University of Helsinki, Finland
Dietmar Pfahl	University of Tartu, Estonia
Rick Rabiser	Johannes Kepler University Linz, Austria
Rudolf Ramler	Software Competence Center Hagenberg, Austria
Andreas Rausch	Technical University Clausthal, Germany
Klaus Schmid	University of Hildesheim, Germany
Rini Van Solingen	Delft University of Technology, The Netherlands
Stefan Wagner	University of Stuttgart, Germany
Dietmar Winkler	Vienna University of Technology, Austria

Sub-reviewers

Michael Brunner, João M. Fernandes, Jan-Peter Ostberg, Nuno Santos, Philipp Zech

Contents

Risk Management and Inspection

Improving the Requirement Engineering Process with Speed-Reviews: An Industrial Case Study

Viktor Pekar[1(✉)], Michael Felderer[1], Ruth Breu[1], Martin Ebner[2], and Albert Winkler[2]

[1] Institute of Computer Science, University of Innsbruck, Innsbruck, Austria
{viktor.pekar,michael.felderer,ruth.breu}@uibk.ac.at
[2] Porsche Informatik, Bergheim, Austria
{martin.ebner,albert.winkler}@porscheinformatik.at

Abstract. Requirement engineering (RE) is a fundamental process in software engineering. RE is based on the software requirement specification (SRS) that contain the systems description. It is a crucial basis for the success of every software project to have understandable and complete SRS. Nevertheless it is a still a challenging task due to the problem that most SRS are written in natural language. Informal textual descriptions involve issues like ambiguity, inconsistency or incompleteness. In this case study we address such issues by introducing a special form of reviews to RE process in an industrial environment. We show improvements in the SRS after the Speed-Reviews started and additionally we present expert-opinions about Speed-Reviews. We conclude that Speed-Reviews are an improvement method that lead to more understandable and less ambiguous SRS.

Keywords: Software engineering · Requirements engineering · Software requirement specifications · Reviews · Inspections · Speed review · Case study

1 Introduction

The requirement engineering (RE) process is an essential part to the success of a medium to large project [1–5]. Software requirement specifications (SRS) are the key artifact in RE [6–8], and therefore are related directly to the overall success of a software project. Incomplete, ambiguous, inconsistent or somehow deficient SRS are even considered as the direct cause for project failures [3,6,8–14]. Still companies have difficulties with SRS and are not satisfied with their results [9].

Consequently, research exists in the field of SRS improving. The IEEE-830 standard (1998) [15] presents a set of criteria that define high quality SRS: Correctness, unambiguity, completeness, consistency, ranking for importance and/or stability, verifiability, modifiability and traceability. Approaches exist, which directly address ambiguity (Nocuous Ambiguity Identification (NAI) [16],

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 3–19, 2015.
DOI: 10.1007/978-3-319-13251-8_1

Collective Intelligence for Detecting Pragmatic Ambiguities [17]), incompleteness (Marama based on EUC-IP [18]) or some other issue. Alternatively, some approaches attend to solve multiple issues at the same time (QuARS [19]). One well-known method-type that addresses several issues at once is the review method.

Related to the IEEE-830 standard [15] criteria set for SRS we classify Speed-Reviews to focus on the following criteria: correctness, unambiguity, completeness and consistency. The term Speed-Review and its concept are inspired by the idea of speed-dating. Speed-dating is based on quick rounds where the attendees switch tables to meet someone new. Each round is limited by a maximum time limit. We perform the case study at one of the largest automobile trade companies in Europe that provides dealer management systems for the international market.

In Sect. 2, we present alternative review methods and explain the Speed-Review procedure detailed in Sect. 3. The case study design and research questions are shown in Sect. 4. Afterwards, we provide our results in Sect. 5 and discuss them in Sect. 6. In the last Sect. 7, we round up the case study and talk about future work insights.

2 Related Work

Even though SRS reviews can be considered commonly as improvement method, which leads to better SRS, we focus on related work only in the area of reviews and no further improvement methods.

Fagan introduced the first basis for reviews in 1976 as a software inspection process [20]. Since then several approaches related to reviews were researched. We distinguish review types by the addressed problem. Above we mentioned that the review methods basically relates to all SRS criteria according to [15]. But it highly depends on the configuration of reviews what SRS problems are in focus. Berling et al. [21] states techniques for reviews as *ad-hoc*, *checklist* and *scenario based*. The *ad-hoc* process does not provide the reviewer with any guidance, whereas the *checklist* is a collection of tasks that has to be considered. The *scenario based* method includes a perspective that the reviewer has to represent, like for instance tester- or customer-perspective. Dependent on the review-method different SRS problems are likely to be addressed. As an example imagine a review with a checklist that states to check for completeness of use cases and artifact-structure. Textual criteria like understandability and ambiguity are likely to be neglected. On contrary, an ad-hoc review will motivate the reviewer to understand the SRS, which automatically involves a higher concentration on informal descriptions, while other criteria lose importance.

A well known method is the Perspective Based Reading (PBR) procedure introduced by Basili et al. [22]. PBR is a method that improves requirement inspections by providing the reviewer with a certain perspective like for instance tester, developer or customer. The reviewer considers the artifact with varying focus on SRS with his role in mind. Based on PBR there exist modification like

a specialized form for teams; the PBRM(T) method [21]. Another example is Problem Driven PBR by Chen et al. [23]. All those methods are related to our Speed-Review approach since the procedure of SRS inspection is addressed. But the boundary conditions for Speed-Reviews consider the whole process and not just the review itself. Actually it would be possible to combine PBR with the Speed-Review method.

Laitenberger et al. [24] present a non-traditional inspection implementation in a case study at Daimler Chrysler. It exceeds the traditional inspections due to restrictions that take place in the case study environment. Inspections differ from reviews since the defect count is primary.

3 Speed-Review Procedure

In order to avoid certain misunderstandings before we define the term Speed-Review, we need to clarify some of the following terms. The term *review* is a generic term that is defined in the International Software Testing Qualification Board (ISTQB) glossary [25] as "an evaluation of a product or project status to ascertain discrepancies from planned results and to recommend improvements." As a result from this definition there exist several more precise review-types like peer- (also known as inspection), formal-, management-, informal- (also known as ad-hoc), technical- and testability-review. In our context the relevant types are informal-reviews and inspections. Informal-reviews are simply not limited by any formal procedure. Inspection is defined by ISTQB as "A type of peer review that relies on visual examination of documents to detect defects, e.g. violations of development standards and non-conformance to higher level documentation. The most formal review technique and therefore always based on a documented procedure".

3.1 Speed Review Description

Based on the previous definitions we now define the term Speed-Review. Our approach is purely addressing the organizational aspect in reviews and leaves flexibility to content-decisions. For instance, it is possible to perform Speed-Reviews informally or based on formal-checklists. Basically, a Speed-Review is a peer-review under specific constraints and limited by a time factor. Before we explain the process in detail, there is need to specify some requirements. First, there needs to be an artifact in form of a document like SRS or target specifications. Second, the artifact needs to be addressable to some responsible person. Usually this will be the author, or someone who becomes responsible for the maintenance of the document.

As the first precondition for a Speed-Review the responsible persons have to choose a number of artifacts that should be reviewed. As an alternative, other colleagues or the project manager might perform this selection. The second precondition before a Speed-Review takes place, is to determine the review partners. We see advantages when authors of similar artifact-types review each other:

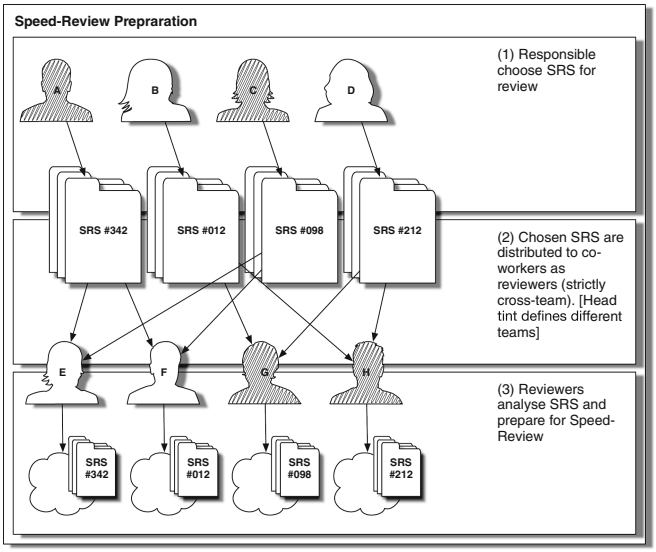

Fig. 1. Speed-Review preparation

(1) It is time efficient. The authors are experienced in, at least, the process where the artifacts take place and might have some degree of know-how related to the domain as well. (2) Authors gain additional insights while reviewing other artifacts and can use such knowledge in their future work. Having reviewers with a similar expertise has the disadvantage of the risk to loose the overview. SRS are a basis for communication with the customer, who has not the expertise as the SRS author. In the end a SRS has to be understandable for all stakeholders.

The method to determine review-partners is highly dependent on the project context and environment size. One option is to do this randomly with certain constraints so that the review-partners vary sufficiently. There are cases where more constraints take place, like in our case study scenario, where other team members should only review multiple teams from different domains. The reason for such a decision was to maximize the information-flow across the department. Information-flow in this context means knowledge about how other colleagues manage SRS.

After review-partners are determined, the last step before the Speed-Review meeting is preparation (see Fig. 1). Every reviewer should study the artifact and take notes. This step should be limited by a maximum amount of time. Obviously, this depends on the average size of the artifacts, but with no limitation there is the danger of a large time effort variety. Besides the time-constraint this step is adequate for deciding what kind of review-type to use. The reviewer might be provided with checklist for the preparation. Speed-Reviews has to be performed regularly. Multiple iterations offer the opportunity to use different approaches for each iteration. As an example for constraints, we look at our case study scenario. The review-partners were determined by a semi-automatic

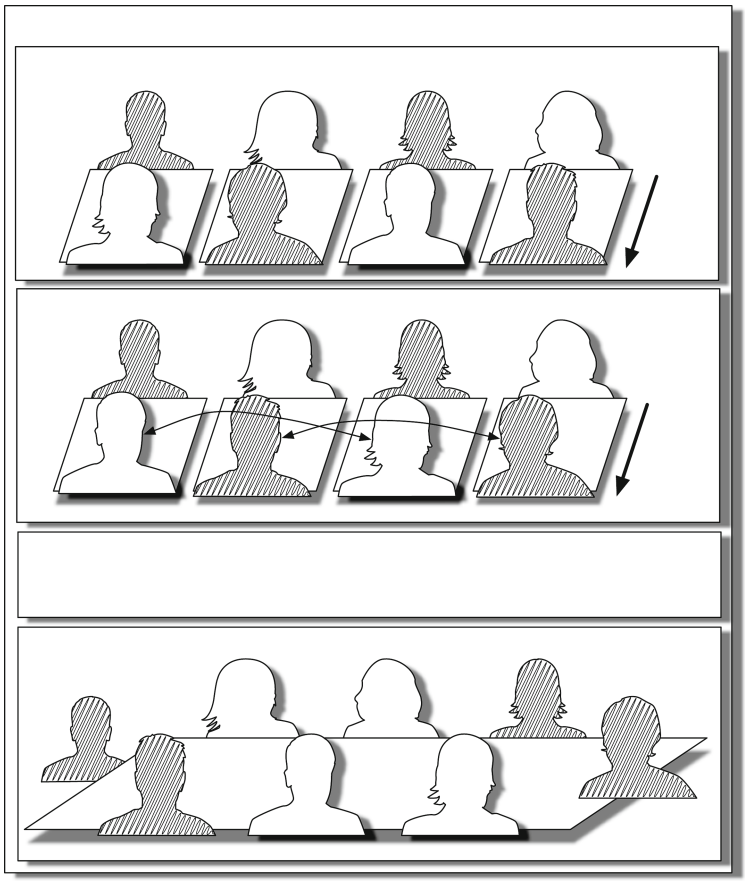

Fig. 2. Speed-Review procedure overview

random procedure. The algorithm considered the cross-team-constraint and offered several constellations of review partners. From this set, some constellations were chosen manually, by assuming interesting review-partners. The selection is performed by the moderator, whose role is explained below.

Once all preparations are finished, the Speed-Review takes place (see Fig. 2). The term Speed-Review is based on the idea of speed dating. This even affects the positioning of the tables in the room where the review-meeting takes place. The review-partners sit in front of each other, with a table in between. The whole meeting is organized in rounds. After an introduction and explanation about the review-procedure, the first round begins and the review-partners take their seats. A round is limited by a specific time limit. This limit is, as well as the time limit for the preparation, dependent on the average artifact size. For our experiments we achieved good results with fifteen-minute time slots. After one round is finished the next takes place and the review-partners change seats according to the new constellation. It proved helpful to have the round-

constellations visible all the time for all attendees. A review-partner constellation per one round means that just one person is giving review-feedback to the other. It is not meant, that both persons give each other feedback at one single round. Obviously, the listener still might interact in form of questions and comparable input. The last part of a Speed-Review takes place after all rounds have been completed. An additional time slot should give all attendees the possibility to give feedback in an open discussion. Preferably the input should address the review procedure or provide ideas for the next iteration.

A crucial factor for a successful Speed-Review is a well-organized moderator. He is responsible for motivating the product owners (PO) to select and prepare the SRS. Furthermore, he sets and coordinates the appointment. Coordination includes the planning of review-partner constellations and paying attention to time limits. Additionally the moderator has the difficult task to analyze the Speed-Review meetings and validate the success. For instance, imagine the case that most review-partners are already done by half of the time limit. The moderator has to notice such occurrences and take countermeasures, which could be a time limit reduction for the next meeting or an extended preparation template.

4 Case Study Design

We use the recommended methodology for case studies provided by Runeson et al. [26].

4.1 Research Questions

The goal of the case study is to show what improvements happen due to Speed-Reviews and how applicable this measure is in practice. For that reason we evaluate the effort for instantiating Speed-Reviews. The following research questions break this goal down into subtasks.

RQ 1. **What effort is needed for establishing and maintaining Speed-Reviews?**
Effort in this context means the time outlay for all participants. This includes the preparation time, the meetings and the organization expenditure for the moderator. Furthermore the difficulty of the related tasks is considered.

RQ 2. **What improvements are performed by Speed-Reviews?**
Improvements are either visible changes in the SRS evolution or expert-opinions, which we evaluated with surveys.

4.2 Context

We perform the case study at Porsche Informatik, one of the largest automobile trade companies in Europe that provides dealer management systems for the international market. The organization consists of approximately 320 employees.

Customer requests and changes lead up to approximately 75 SRS per release. One release lasts for three month and a project related to a SRS can have a varying effort in a range of ten to a thousand hours. The process from a client request to the SRS consists of multiple steps, which are justified by the history of the organization. We neglect the steps between the client request to the SRS, since the process in between is not relevant for the case study.

The organization exerts agile methods, mainly Scrum, even though the overall RE process is still based on the Waterfall-model. A SRS is passed to a PO who is responsible for managing it. The SRS is the basis for future communication with the customer and has contractual value. The POs are separated in three different teams, based on business expertise. This fact is important, because different teams might process multiple requests of the same client. The teams are assigned to *service*, *sales* and *internal base* domains. For us it is important to know that expertise varies across teams.

The process of creating SRS based on the client request is subject to the responsible PO and strongly intuitive. Beside an editor template, that is purely optional, there were no guidelines for this process step at the time we performed the case study. The editor template was an attempt to maintain sections for a new SRS and reach an uniform structure. The template was never introduced properly nor reached a satisfying state, according to the POs statements.

We observed a range of documents that are described in much detail using primarily natural language and on the other hand SRS with very brief descriptions in bullet-list format. The cause for this variety in SRS is the POs unequal expertise and experience.

This lack of correspondent unity between SRS has following drawbacks: (1) Customer and (2) internal colleagues cannot rely on an expected artifact structure. In case of (1) this leads to an increased workload for the customer to break into the SRS. Additionally, the communication effort between customer and PO is likely to increase. The drawback for (2) is of course an increased time expense when working with varying document structures as well. Internal colleagues are all persons that contribute or use the document. This includes the programmers as well as the head of department. We observed the circumstance where colleagues avoided reading the SRS and instead directly contacted the responsible author for explanation. Obviously, such a detour does not scale and we do not consider this as a solution for non-uniform SRS.

We target the mentioned issues with the introduction of Speed-Reviews. In this work we show how the review procedure is performed at Porsche Informatik and how we documented parts of the RE process before and after.

4.3 Data Collection Procedure

As the very first step we analyzed key issues in SRS by interviewing the POs. Those interviews were organized as one-on-one sessions that took half an hour each. The procedure was a combination of a predefined questionnaire (see Table 1) and ad-hoc interview. The reason for this hybrid-approach was to give the POs the opportunity to express their perspective. Nevertheless some key questions were discussed with each person. You find an overview of the questions in the

appendix Sect. 1. Furthermore the POs stated problems that overlapped across the interviews. Resulting from the information we gathered with the interviews from the previous step, we decided to establish reviews as a first measure (see Tables 2, 3 and 4).

According to the small number of POs the reviews were performed as Speed-Reviews. Speed-Reviews are a newly defined concept, which we explained in detail in Sect. 3. Before the Speed-Review meeting each PO had to prepare two reviews of a SRS, with a maximum effort of fifteen minutes each. Additionally everyone had to choose two SRS from under her own responsibility. The constellation about who reviews whom, was calculated by a random procedure and controlled by the moderator. In this specific scenario it was a requirement that POs from the same team would not review each other. With cross-team pairing only the efficiency of experience exchange was intended to be increased. While the Speed-Review was taking place, two transcribers where switching between the tables and writing down the most important statements. Most statements addressed the problems we identified in the one-on-one interviews. The Speed-Review meeting was completed with an open discussion with the intention to gather feedback on the Speed-Review experiment. The transcribers logged the POs responses.

Besides the qualitative investigation, we conduct analysis on the SRS database throughout the process. Due to technical restrictions this analysis has to be performed manually. We have access to historical SRS that were created before the initiation of Speed-Reviews. Therefore, we compare SRS from before and after the first Speed-Review took place. We chose two requirements from before and after the Speed-Review beginning of each PO. That leads to an analysis of 24 SRS in whole. The reason why we compare old with new SRS and not the evolution of single SRS during time, is the restriction that existing SRS has not be reworked according to Speed-Review findings. In the Introduction we state that Speed-Reviews attempt to improve correctness, unambiguity, completeness and consistency. It requires expertise in the domain of Dealer Management Systems to analyze correctness, which is why we have to limit our comparison to unambiguity, completeness and consistency. The authors from University of Innsbruck and not employees from Porsche Informatik selected the analyzed SRS. This decision reduces the risk of POs selecting their favorite SRS and therefore reducing objectivity of the analysis.

As last step for evaluation we performed a survey including all eight POs. The goal was to question usefulness and deploy-ability of Speed-Reviews. The survey uses a seven-step Likert-Scale with an intuitive range like in Fig. 3. The interviewed persons had to mark a position on the scale depending on their opinion. We explicitly did not offer a neutral-option since all questions were

Fig. 3. Sample for used Likert-Scala in the Product Owner Survey

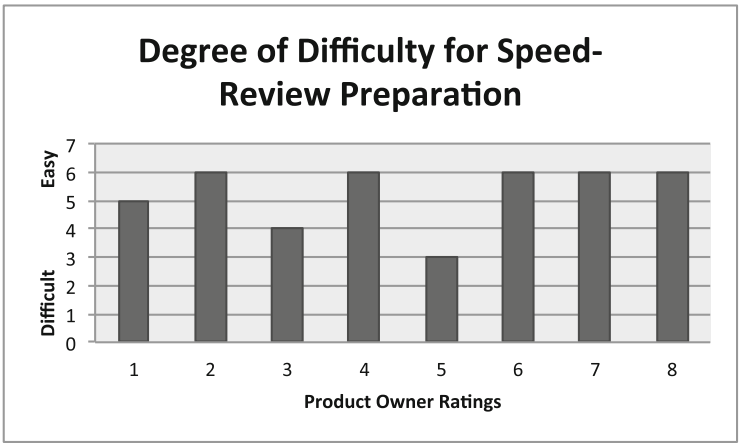

Fig. 4. Rating distribution related to difficulty for Speed-Review preparations

answerable for all POs. All POs performed the survey at once in the same room and anonymously.

5 Results

In this section we present our results. We distinguish between the evaluation based on expert opinions considering the Speed-Review process and visible changes in the SRS artifacts.

5.1 Speed-Review Process Evaluation

The questionnaire that was handed out to the POs after several Speed Reviews meetings. The full questionnaire is available in Table 5 but in the following we present only the results that are mostly relevant to the research questions.

In Fig. 4 we see the results related to the opinion of the POs about the review-preparation difficulty. Besides one neutral rating all surveyed persons rated the preparation as quite easy. Out of eight POs five even considered the preparation as almost very easy. The average rating is *5,25* with *7* having the meaning *most easy*.

Besides the difficulty for the review preparation we asked about the effort for the Speed-Review process. This includes the preparation time as well as the Speed-Review meetings. In Fig. 5 we show the distribution for the single ratings. As average the POs considered this effort as *low*, with *2,13* where *1* represents the lowest effort in the scale. The results about preparation-difficulty and Speed-Review effort are related to the first research question about overall-effort when performing Speed-Reviews.

In Fig. 6 we present two layers of an improvement degree. This degree relates to the SRS-writing-skills of the POs. We asked two separate questions whether

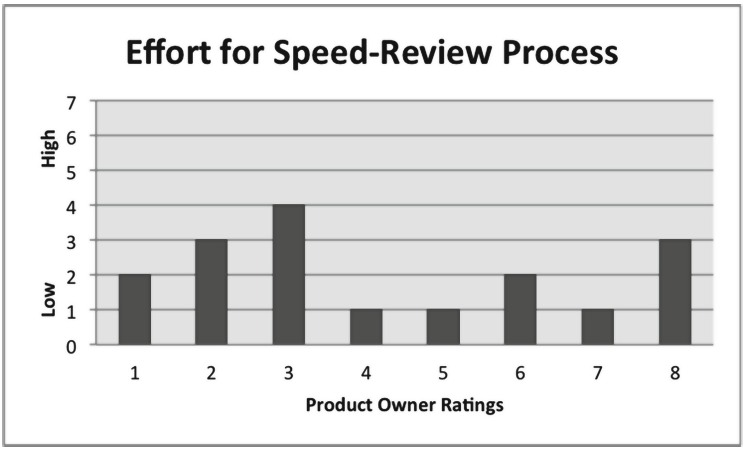

Fig. 5. Rating distribution related to effort for the Speed-Review process

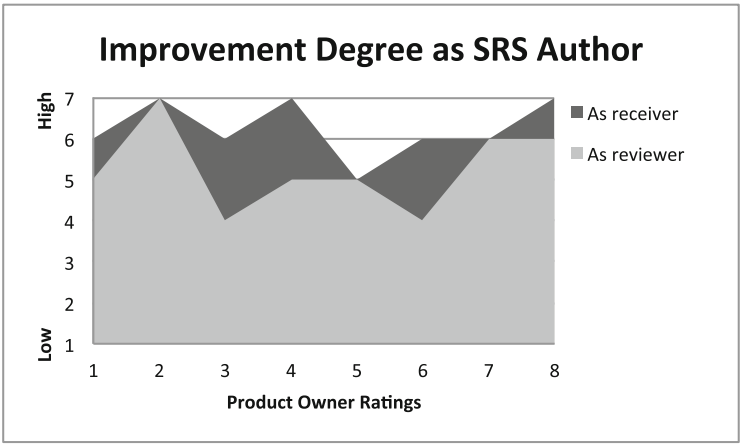

Fig. 6. Improvement Degree as SRS author

the interviewed person improved as SRS author either by receiving or providing feedback. Averagely the surveyed audience considered the receive of feedback (*5,25*) less productive than providing it (*6,25*; while 7 is the most possible improvement degree on scale). Three cases indicated an equal improvement degree for both methods.

For one question the POs estimated the durability of Speed-Reviews in a long term. At the time the survey took place, Speed-Reviews still were new and so it was hard to predict its position in long term. Nevertheless, the feedback is highly positive with five out of eight POs estimations for a very long durability. The average rating is *6,38* with no estimation that predicts a short durability. For details check Fig. 7.

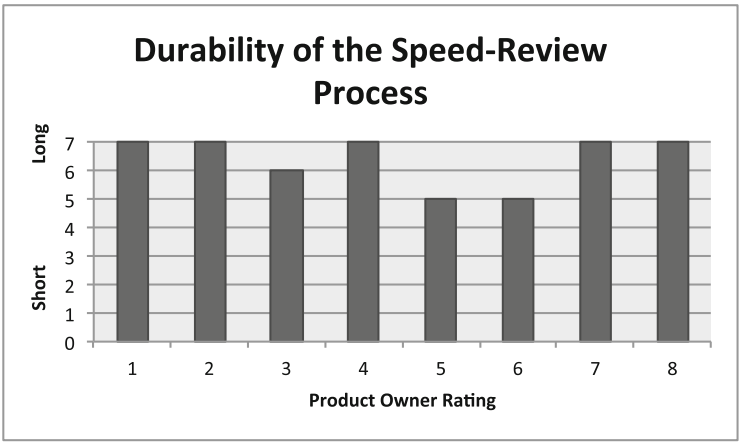

Fig. 7. Durability of Speed-Reviews in long term

5.2 Software Requirement Specification Results

In this Section, we present the second part of our results that are based on our observations on the SRS evolution. In Sect. 4.3 we explain why we choose 12 (*old*) SRS that were created before the Speed-Review initiation and 12 (*new*) SRS that were started afterwards. First, we explain the criteria for the comparison procedure. These are based on our findings from the interviews with the POs: (1) uniformity of structure, (2) readability and (3) understandability. In Sect. 4.3 we stated that we also consider the IEEE-830-1998 [15] criteria unambiguity, completeness and consistency. Because of the size and varying complexity of the SRS we cannot analyze the artifacts in full detail. While analyzing a SRS we ask the following questions: (1) Is a short overview provided? (2) Has the artifact a complexity hierarchy that leads the reader from an abstract level to more details? (3) Has the artifact a structure at all? Structured in this context means that recommended procedures were followed. (4) Is the basic SRS idea understandable by a non-professional reader? Already at the first Speed-Review the POs decided that there is need for an abstract that provides a short overview about the SRS. None of the old analyzed SRS showed an abstract. Of 12 new considered SRS, 11 provided such a short description. We analyze, whether or not the SRS has a complexity hierarchy. It means that technical explanations or use cases appear in lower sections while generic descriptions introduce the whole SRS. Of the old SRS, 7 show no hierarchy and 5 approaches with shortages. The new SRS consist of 9 that have a clear complexity hierarchy, 1 SRS with shortages and 2 that still miss this criterion. A shortage can be for instance a missing information gap between the abstract and the use cases. With no proper business process explanation it still can be difficult for the reader to understand technical use cases. The new SRS, which were analyzed compose from 9 hierarchically structured, 2 unstructured and 1 partially structured artifacts.

Additionally we regard the structure without a focus on complexity. For instance recommended sections are missing or copied conversations compensate for explanations. The old SRS show 1 structured, 8 partially and 3 unstructured formats. The improvements we observed with new SRS are at 9 structured and 3 partially structured SRS. Both criteria related to structure are overlapping, since a hierarchical complexity structure can only exist with a generic structure. In Sect. 3, we explain that SRS are the basis for communicating with the customer. Therefore, it is an important criterion that reader with no specialized expertise can at least understand the rudiments of every SRS. The authors of this case study that are no employees at Porsche Informatik, have the perfect lack of specialized expertise to analyze the SRS from customer-perspective. For the case that we are able to understand the rudiments in less than 5 min, we consider the SRS as understandable for non-professionals. It was hard to get an understanding with old SRS: 8 are rated as not understandable, 2 partially and 2 as fully understandable. Due to the abstract and improved structure, the results for new SRS are: 9 SRS fully, 1 partially and 2 not understandable.

6 Discussion

In the following, we discuss whether the research questions are answered in this work. The effort that is needed for establishing Speed-Reviews is stated in Sect. 3. The exact effort value is dependent on the SRS size, number of assigned persons and on the variable settings (like maximum preparation time limit). We surveyed the POs to determine whether the extra effort is justified and the feedback was positive without exception. After introducing Speed-Reviews at Porsche Informatik we quickly recognized more uniform structured SRS with a better understandability compared to old SRS.

We observe that new SRS become more similar to each other in terms of structure. Non-uniform SRS is one major problem and therefore, this change is a success. We explain this changes by triangulation based on both evaluation methods. For the SRS comparison it is obvious since the structure in SRS simply becomes visible for the reader. POs stated during the survey that a better coordination of the SRS creation process is given due to Speed-Reviews, which leads to structure alignment. Better coordination occurs because of review templates that show the PO how to analyze SRS but as well how to write them. Additionally, the knowledge exchange across employees and teams increases the degree of coordination and leads to better group dynamics. Besides structure and uniformity we experienced an improvement in understandability and readability. Some old SRS did not provide any introduction, which makes it hard and time-consuming to understand even the basics of a SRS. Due to Speed-Reviews the POs identified the need for an abstract. Our results from Sect. 5.2 verify the success of this decision.

Quantitative measurements that show improvements to SRS would be a valuable complement. Such a data collection might be considered in future projects.

6.1 Threats to Validity

The greatest threat to validity is that visible improvements in SRS after the Speed-Review launch, might have other causes than the reviews itself. For instance, the POs were provided with highly technical checklists for the SRS creating-process during the time of Speed-Review introduction. Those lists remind the PO for example to consider dependencies to other systems. We do not believe this bias our results but nevertheless, to guarantee a better reliability of the case study we decided to perform the qualitative evaluation based on PO opinions additionally to the SRS comparison.

A threat that directly affects the SRS comparison is the varying technical complexity of SRS. Especially for criteria like understandability and readability it is obvious that simple SRS are easier to understand than complex SRS. To ease this risk we selected SRS with similar effort estimations. Still the threat remains because complexity of a topic does not need to be only size-related.

A third threat related to SRS analysis is the lack of formality in the comparison procedure. The varying structure of the SRS, the use of natural language and non-uniformity between SRS prohibited formal methods. Therefore the authors performed the SRS comparison informally, which might include a certain degree of subjectivity.

Furthermore, we are not able to compare the Speed-Review method to other review-methods. We show that Speed-Review bring improvements to the RE process. But how these effects would differ from alternative methods is out of scope, since no review measures existed beforehand.

7 Conclusions and Future Work

First, we explained the importance of SRS and the term *review* in Sect. 1. In Sect. 2 we gave an overview about related review-approaches and finally explained Speed-Reviews in detail in Sect. 3. We introduce the Speed-Review procedure at Porsche Informatik and present the results in a case study beginning in Sect. 4. The actual results are shown in Sect. 5 and discussed in Sect. 6.

We defined the Speed-Review procedure and successfully established it at Porsche Informatik. The improvements are evaluated by expert opinions of the POs who are the responsible persons for SRS. Additionally, we analyzed the evolution of SRS. For that we compared artifacts that were initiated before Speed-Reviews with those that were created afterwards. Speed-Reviews lead to more structured, understandable and readable SRSs.

It is yet to be evaluated whether Speed-Review improvements only appear in short-term or also in long-term. We plan to observe the Speed-Review evolution at Porsche Informatik with the future goal to answer this question. Furthermore, the Speed-Review technique is in a prototype state. In practice, we already experienced shortages like long time limits, which lead to concentration loss of the participants. Besides that the POs mentioned the risk when the person who is to be reviewed is the selector of the artifacts at the same time. It is necessary to test Speed-Reviews for longer time-periods and to refine the method.

A Interview Guidelines and Questionnaires

Table 1. Interview guidelines for product owners

Significance for Target Specifications
How important do you consider the target specification quality for the overall success of a project?
How important do you consider the completeness of target specifications?
How important do you consider the understandability of target specifications?
How important do you consider the severity of ambiguity in target specifications?
How important do you consider a uniform structure across all target specifications?
Target Specification Problems
How effective do you consider the requirements engineering process at you company?
How effective do you consider the used tooling?
Do you think the quality of an end product could be increased by improving the requirement engineering process?
Did you have problem understanding target specifications? (If yes - why?)
Did you experienced misunderstandings about term definitions or processes? Or any misunderstandings related to ambiguity?
If you are the initiating author of a target specification - were you always certain where to start?
Can you recall other issues related to target specifications that were not mentioned yet?

Table 2. Final Questionnaire for Product Owners related to Review-Preparation

Evaluation for the preparation	
How much time did you approximately spent for review preparation? (in minutes) *[Keep in mind that in this case study an recommendation of an maximum of 20 min was given]*	24,38
Did you consider the preparation as easy?	2,13
Did you feel limited by the maximum time limit you should spend for the preparation?	5,25
Did you prefer to have presets for the review-preparation or the ad-hoc approach?	3,38
Did you started to re-think your artifacts while reading others?	5,38

Table 3. Final Questionnaire for Product Owners related to the Speed-Review process

Evaluation for the Speed-Review meeting	
Did you find the organization of the Speed-Review intuitive from the beginning?	6,75
Did you consider the dialog maximum time as sufficient?	6,88
Was your feedback accepted by your review-partner?	6,88
Did you received constructive feedback?	6,63
Consider the feedback you received from different review-partners. Was it similar?	4,63

Table 4. Final Questionnaire for Product Owners related an overall estimation about Speed-Reviews

Evaluation for the Speed-Review process	
Consider the overall effort that you experienced for one review iteration (including preparation and meeting time). How do you rate the ratio between effort and benefit?	6,13
Did you improved as author due to receiving feedback?	6,25
Did you improved as author due to performing reviews?	5,25
Would you consider Speed-Reviewing as a sustained improvement method?	6,38
How much are Speed-Reviews dependent to the organization procedures?	3,75

Table 5. Final Questionnaire for Product Owners related to key statements

Evaluation for the Speed-Review Process Ideas
Please name advantages that appear due to Speed-Review usage
Can you come up with disadvantages that appear after Speed-Review execution?
Where do you sense weaknesses in Speed-Reviews?
Please provide improvement measures if possible

References

1. Bucchiarone, A., Gnesi, S., Pierini, P.: Quality analysis of NL requirements: an industrial case study. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp. 390–394, Aug 2005
2. Ormandjieva, O., Hussain, I., Kosseim, L.: Toward a text classification system for the quality assessment of software requirements written in natural language. In: Fourth International Workshop on Software Quality Assurance: In Conjunction with the 6th ESEC/FSE Joint Meeting, SOQUA '07, pp. 39–45. ACM, New York (2007). http://doi.acm.org/10.1145/1295074.1295082

3. Scheffczyk, J., Borghoff, U., Birk, A., Siedersleben, J.: Pragmatic consistency management in industrial requirements specifications. In: Third IEEE International Conference on Software Engineering and Formal Methods, SEFM 2005, pp. 272–281, Sept 2005
4. Hu, H., Zhang, L., Ye, C.: Semantic-based requirements analysis and verification. In: 2010 International Conference on Electronics and Information Engineering (ICEIE), vol. 1, pp. V1-241–V1-246, Aug 2010
5. Tjong, S., Hallam, N., Hartley, M.: Improving the quality of natural language requirements specifications through natural language requirements patterns. In: The Sixth IEEE International Conference on Computer and Information Technology, CIT '06, pp. 199–199, Sept 2006
6. Belfo, F.: People, organizational and technological dimensions of software requirements specification. In: 4th Conference of ENTERprise Information Systems Aligning Technology, Organizations and People (CENTERIS 2012), vol. 5, pp. 310–318. http://www.sciencedirect.com/science/article/pii/S2212017312004653
7. Mat Jani, H., Tariqul Islam, A.: A framework of software requirements quality analysis system using case-based reasoning and neural network. In: 2012 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM), pp. 152–157, Oct 2012
8. Heck, P., Parviainen, P.: Experiences on analysis of requirements quality. In: The Third International Conference on Software Engineering Advances, ICSEA '08, pp. 367–372, Oct 2008
9. Gross, A., Doerr, J.: What you need is what you get!: the vision of view-based requirements specifications. In: 20th IEEE International Requirements Engineering Conference (RE), pp. 171–180, Sept 2012
10. Saito, S., Takeuchi, M., Hiraoka, M., Kitani, T., Aoyama, M.: Requirements clinic: third party inspection methodology and practice for improving the quality of software requirements specifications. In: 2013 21st IEEE International Requirements Engineering Conference (RE), pp. 290–295, July 2013
11. Abernethy, K., Kelly, J., Sobel, A., Kiper, J., Powell, J.: Technology transfer issues for formal methods of software specification. In: Proceedings of the 13th Conference on Software Engineering Education amp; Training, pp. 23–31, Mar 2000
12. Polpinij, J.: An ontology-based text processing approach for simplifying ambiguity of requirement specifications. In: IEEE Asia-Pacific Services Computing Conference, APSCC 2009, pp. 219–226, Dec 2009
13. Huertas, C., Jurez-Ramrez, R.: NLARE, a natural language processing tool for automatic requirements evaluation. In: Proceedings of the CUBE International Information Technology Conference, CUBE '12, pp. 371–378. ACM, New York (2012). http://doi.acm.org/10.1145/2381716.2381786
14. Salger, F., Engels, G., Hofmann, A.: Inspection effectiveness for different quality attributes of software requirement specifications: an industrial case study. In: ICSE Workshop on Software Quality, WOSQ '09, pp. 15–21, May 2009
15. IEEE, Ieee recommended practice for software requirements specifications. Institute of Electrical and Electronics Engineers (1998)
16. Yang, H., Willis, A., De Roeck, A., Nuseibeh, B.: Automatic detection of nocuous coordination ambiguities in natural language requirements. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10, pp. 53–62. ACM, New York (2010). http://doi.acm.org/10.1145/1858996.1859007

17. Ferrari, A., Gnesi, S.: Using collective intelligence to detect pragmatic ambiguities. In: 20th IEEE International Requirements Engineering Conference (RE), pp. 191–200, Sept 2012
18. Kamalrudin, M., Hosking, J., Grundy, J.: Improving requirements quality using essential use case interaction patterns. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pp. 531–540. ACM, New York (2011). http://doi.acm.org/10.1145/1985793.1985866
19. Bucchiarone, A., Gnesi, S., Fantechi, A., Trentanni, G.: An experience in using a tool for evaluating a large set of natural language requirements. In: Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, pp. 281–286. ACM, New York (2010). http://doi.acm.org/10.1145/1774088.1774148
20. Fagan, M.E.: Design and code inspections to reduce errors in program development. In: Broy, M., Denert, E. (eds.) Pioneers and Their Contributions to Software Engineering, pp. 301–334. Springer, Heidelberg (2001)
21. Berling, T., Runeson, P.: Evaluation of a perspective based review method applied in an industrial setting. IEE Proc. Softw. **150**(3), 177–184 (2003)
22. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The empirical investigation of perspective-based reading. Empirical Softw. Eng. **1**(2), 133–164 (1996)
23. Chen, T., Poon, P.-L., Tang, S.-F., Tse, T. H., Yu, Y.: Towards a problem-driven approach to perspective-based reading. In: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering, pp. 221–229 (2002)
24. Laitenberger, O., Beil, T., Schwinn, T.: An industrial case study to examine a non-traditional inspection implementation for requirements specifications. Empirical Softw. Eng. **7**(4), 345–374 (2002). http://link.springer.com/article/10.1023/A:1020519222014
25. I. ISTQB, Glossary of Testing Terms (2012)
26. Runeson, P., Hst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. **14**(2), 131–164 (2009). http://link.springer.com/article/10.1007/s10664-008-9102-8

Towards a Perspective-Based Usage of Mobile Failure Patterns to Focus Quality Assurance

Konstantin Holl[1(✉)], Frank Elberzhager[1], and Vaninha Vieira[2]

[1] Information Systems Quality Assurance, Fraunhofer Institute for Experimental Software Engineering IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{konstantin.holl, frank.elberzhager}@iese.fraunhofer.de

[2] Computer Science Department, Fraunhofer Project Center for Software and Systems Engineering at UFBA, Federal University of Bahia, Av. Ademar de Barros, 500, Ondina, Salvador-BA 40110-170, Brazil
vaninha@dcc.ufba.br

Abstract. The use of mobile applications for business tasks calls for effective quality assurance during development to prevent potential failures of the mobile application and the consequential costs. Essential activities of quality assurance are to inspect the requirements specification and to test the realized mobile application. Both activities ideally benefit from the knowledge of typical failure patterns in order to guide quality assurance engineers and to focus the quality assurance, i.e., to direct the attention of quality assurance on these failure patterns. For this purpose, a mobile-specific failure pattern classification was derived in previous work. In this paper, we introduce an initial quality assurance approach, which considers inspection and testing in a combined way and methodically uses the classification within the mobile context. The developed method FIT4Apps, which is based on the flexible, efficient reading technique perspective-based reading, allows purposeful use of the proposed failure pattern classification. As proof of concept, we developed a tool prototype, which generates basic perspective-based scenarios considering the derived failure pattern classification in order to support inspection and testing activities.

Keywords: Quality assurance · Mobile testing · Inspection · Perspective-based reading · Failure pattern classification

1 Introduction

The market for mobile devices is rapidly growing [1]. Furthermore, new mobile applications are continuously developed and shipped. Especially in the context of mobile business applications, several advantages are expected, e.g., higher availability, quick distribution of new information, and faster reaction time. However, as it is true for software development in general, a high quality of such applications is a necessity due to the risk of severe consequences, such as lost revenue or reduced efficiency. For example, consider a mobile application for production process control, which offers (i) information about the current status for monitoring the production, and (ii) decision support, like the collection and interpretation of environment parameters. An application failure

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 20–31, 2015.
DOI: 10.1007/978-3-319-13251-8_2

could lead to wrong decisions or unintended production effects, controlled by the mobile application. In this example, the usual consequences are very costly due to the high run-up time of production processes.

During the last decades, several quality assurance methods, techniques and tools were developed and are usually applied during software development. For instance, different inspection [2] and testing techniques [3] support, e.g., quality assurance engineers, inspectors and testers, to ensure the high quality of a software. However, as the mobile trend is relatively new, such quality assurance techniques are barely adapted to the new needs and challenges, such as:

- Mobile applications are developed in short, often agile, development cycles, and a fast time to market is expected [4].
- The set of requirements as the basis for the implementation and also for testing is changing during the development [5].
- Quality assurance is often unfocused due to unknown failure patterns in the mobile domain [6].

Consequently, compared to traditional quality assurance, mobile quality assurance has to be adapted to these new challenges by an approach, which aims eventually at being more efficient to be suitable for short development cycles, and being more effective by coping with a changing test basis and by focusing on typical failure patterns. Failure patterns represent the relation of fault aspects and their corresponding failures.

In an earlier publication [7], we presented how we derived a mobile-specific failure pattern classification. In this paper, we show a combined quality assurance method, which uses the failure classification. Furthermore, we demonstrate our method partially via a proof-of-concept using a tool prototype. The research question we address is the following:

> RQ: How to use the derived mobile-specific failure pattern classification to focus quality assurance of mobile applications?

The structure of the paper is as follows: Sect. 2 describes the related work, where we provide an overview of quality assurance aspects regarding mobile applications. Furthermore, the section explains the failure pattern classification, which can be used to focus quality assurance, i.e., to direct the attention of quality assurance to these failure patterns. Section 3 presents our combined quality assurance method, which is tailored for mobile development. It also describes a detailed example, our tool prototype, and a discussion regarding the overall approach. Finally, Sect. 4 concludes our paper and gives an outlook on future work.

2 Related Work

The related work is divided into a general mobile quality assurance part, which describes recent publications about issues in this field, and into an outline of the recently developed failure pattern classification, which is fundamental to lead over to the development of the method to focus quality assurance of mobile applications.

2.1 Quality Assurance of Mobile Applications

Launching a mobile application development project often leads to questions regarding how the methods, techniques, and tools differ from classical software development projects, e.g., in the field of desktop applications. Methodological questions arise regarding the peculiarities of aspects, such as the application's architecture and security, and phases like requirements engineering and quality assurance. Those questions can usually be answered by using established approaches for non-mobile development projects. Also, existing tools are basically the same used in classical software development projects. For instance, recording and playback tools, which are popular in the area of mobile applications, are based on the already established concepts.

Differences in developing and using mobile applications are often related to technological aspects. The main peculiarities of applications on mobile devices such as smartphones and tablets are, according to Muccini et al. [8]:

- limited resources (e.g., battery)
- user interface (e.g., touchscreen)
- context awareness (e.g., mobile connectivity)
- diversity (e.g., devices and operating systems).

Muccini et al. [8] answered and confirmed their research questions, whether mobile applications are different from traditional ones and whether they require different and specialized new testing techniques, by discussing the peculiarities of mobile applications. They concluded with regard to mobile applications testing that there are many challenges "related to the contextual and mobility nature of mobile applications" and furthermore that "performance, security, reliability, and energy are strongly affected by the variability of the environment where the mobile device moves toward".

Dantas et al. [6] considered mobile applications so specific that the derivation of testing requirements is necessary to enable efficacious and productive quality assurance. One of the conclusions drawn from the conducted interviews was that "most companies do not have a specific testing process for mobile applications" and that "tests are not executed systematically, and are defined based on previous experience or intuition of developers and testers". Furthermore, it is possible to achieve effective quality assurance in spite of the given challenges by helping "the testers to find specific errors on mobile environment".

Franke and Weise [9] claimed that the challenges of mobile applications development "make great demands on software and ask for specific approaches and methods for quality assurance" and that quality assurance methods "vary between different kinds of software, in particular between software for mobile and desktop applications". Their contribution describes the provision of a software quality framework based on finding and defining key qualities of mobile applications.

Overall, existing methods and insights regarding quality assurance in mobile application development projects – which are typically conducted using ad hoc, classical (e.g., regression testing within the waterfall model), or agile state-of-the-art approaches (e.g., test first within Scrum) [4] – have increased in importance as a key discipline, but lacks specialized methods, expertise, and environment [10] and are often not effective due to the missing focus on the peculiarities of mobile applications.

2.2 Focus Quality Assurance on Mobile-Specific Failure Patterns

Due to the problem that there is no established information about typical mobile application failure patterns, our previous contribution [7] described how to collect and to classify frequently mentioned fault aspects of mobile application failures. Furthermore, approaches for creating classifications of defects respectively failures found in the literature were described like the schemes of Li et al. [11] and Mauser et al. [12], which were encouraged, among others, by the ODC [13] and the IEEE Standard Classification for Software Anomalies [14]. Considering the experiences, made in multiple mobile application development projects, the previous contribution explained and classified types of typical mobile application faults and failures.

The project experience comprised four mobile applications, which have been developed within six months, in 2013. The quality assurance team analyzed the failure reports of this project with the intention to create a classification for failures and typical fault aspects. In our previous contribution [7], we used the term *fault* as the origin of a failure, and the term *fault aspect* as the focus of a test case that leads to a specific failure of the mobile application, which is called fault class (as part of the failure pattern classification) in the current contribution.

The state of the art regarding typical mobile failures and typical fault classes was captured based on publications by a literature review according to Kitchenham [15]. The review was done for the time span between the year 2006 and the contribution date in 2014, what resulted in 26 publications, which could be identified to support the development of the classification (see Fig. 1).

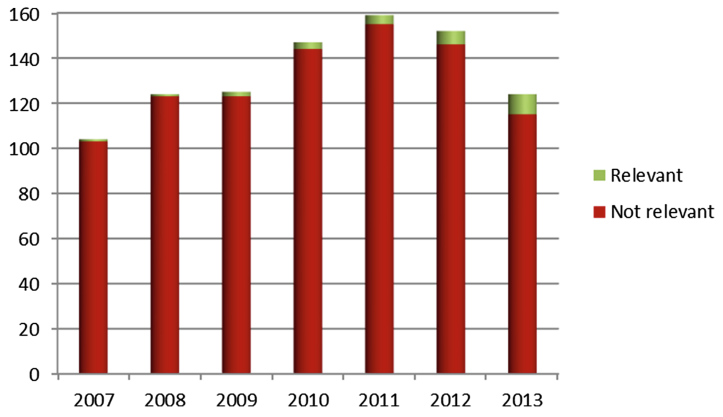

Fig. 1. Distribution of publications containing related work

In our earlier publication, the fundamental research question "How to classify detected mobile-specific failures?" was answered based on project experiences and related work regarding failure classifications. The ensuing research question "Which typical fault aspects exist and how to integrate them into the failure classification?" was answered based on project experiences and the literature review regarding mobile failures [7]. We concluded that the "work on both research questions resulted in overlapping

information on the one hand and complementary information on the other hand. Combination with the knowledge of other classification schemes enabled the creation of a basic mobile-specific failure classification including categorized typical fault aspects […]. This classification could be successfully evaluated and optimized by applying it to multiple mobile application developments." [7] Due to the defined relations between failures and fault aspects, we call this classification for short: failure pattern classification.

3 A Method to Focus Quality Assurance of Mobile Applications

The creation of the failure pattern classification led to the question how to use it within quality assurance. This section explains the developed method, gives an example of its application and describes a first prototype for a partial automation.

3.1 Description of the FIT4Apps Method

A quality assurance method that addresses the main reasons for the insufficiency of common approaches can be approached by combining testing activities with an inspection technique, both influenced by the mobile-specific failure pattern classification. Therefore, the method FIT4Apps (Focus Inspections and Tests for Mobile Applications) was developed.

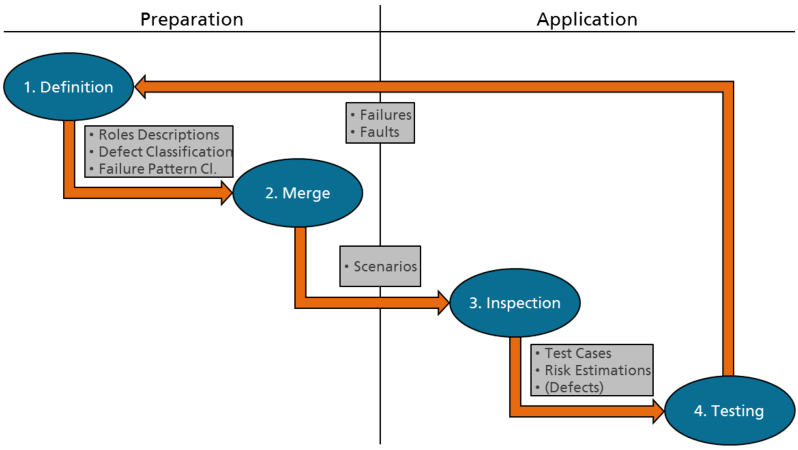

Fig. 2. The process model of the FIT4Apps method.

FIT4Apps contains two preparation steps, *Definition* and *Merge*, which enable the application of the method: *Inspection* and *Testing* (see Fig. 2).

1. Definition. The method's first step is the definition of descriptions of roles, which are taking part in the development, the defect classification regarding potential deficiencies of the requirements specification, and the failure pattern classification. The definition comprises also the relations between these artifacts. The classified **Failure Patterns** are structured by:

- **Failure Sections** (*Behavior*, *Design* and *Content*), which categorize the
- **Failure Classes** (e.g., *Interaction Element*, *Transition*, and *Static Text*), which comprise by their entities (*wrong*, *missing* and *extra)* aspects "in which a system or system component does not perform a required function within specified limits" [14]. The failure classes are related to
- **Fault Classes** (e.g., *Network Disconnect*, *Low Battery*, and *Screen Orientation*), which comprise typical fault aspects [7] and belong to several
- **Fault Sections** (e.g., *Connection*, *Energy*, and *User Interface*) for the purpose of categorization.

Furthermore, the **Failure Patterns** are related to:

- **Roles** (*Tester*, *Designer*, and *User*), which are each qualified for the detection of certain failure patterns and defects of certain
- **Defect Classes** (*Missing Information*, *Ambiguous Information*, *Extraneous Information*, *Inconsistent Information* and *Incorrect Facts*), which comprise potential deficiencies of requirements specifications [16].

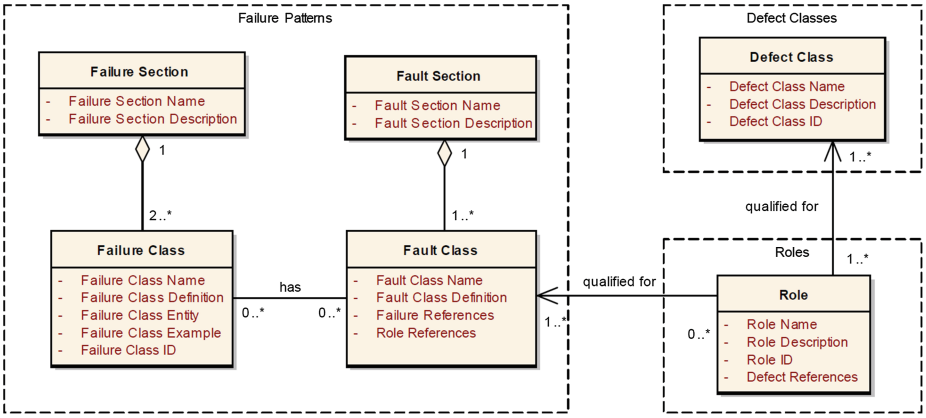

Fig. 3. Relation between failure patterns, roles, and defect classes.

The failure pattern classification contains three failure sections, which were split into nine failure classes. Each failure class has three entities, which were related to a maximum of 4 of 21 defined fault classes. Overall, 44 failure patterns (i.e., typical relations between failure classes and fault classes) were defined [7]. As part of the current contribution, the failure pattern classification was related to the typical roles and defect classes to enable a perspective-based usage of the derived classification (see Fig. 3).

2. Merge. The defined artifacts of the first step are merged to perspective-based reading (PBR) scenarios structured according to Shull et al. [17]. This represents a convenient reading technique for inspecting the requirements specification in the mobile context, due to its efficiency and flexibility [17]. Each scenario contains:

- an introduction, to understand and, if necessary, to adopt the characteristics of a special role,
- instructions, which describe the tasks the reader has to fulfill while inspecting the requirements specification,
- and questions, which indicate aspects of the requirements specification to be checked by the reader concerning the defect classes and failure patterns.

The merge occurs based on the defined relations. For instances: the *tester* is qualified to detect *missing information* in the requirements specification; a *changing orientation* of the mobile device can lead to a *missing interaction element* in the user interface.

3. Inspection. As first step of applying the method, the inspector of each perspective reads every PBR scenario belonging to his role. The definition of typical roles taking part in the project is intended to support the inspection due to the separation of the points of view of the different inspectors. This intention aims at less overlapping results regarding the findings. Typical roles would be, for instance: a user, who is determined to use the mobile application after it has been fully developed, marketed, and installed; or a tester, who operates the mobile application under specified conditions, observing and recording the results and evaluating some aspects of the mobile application. Inspectors of every role are instructed to perform perspective-specific tasks. This means in case of the user's perspective, to conduct risk estimations with focus on given failure patterns. In case of the tester's perspective, it means to derive test cases with focus on given failure patterns. Furthermore, every role has to answer perspective-specific questions to find defects in the requirements specification while following the instructions. The motive of this technique is, to perform an inspection in parallel to those tasks, which have to be performed anyway by this role.

4. Testing. As last step of the method, the derived test cases, which have got the focus on the mobile-specific failure patterns, are composed to a test suite which is part of the development project's test plan. According to the test plan's strategy, the test cases are executed. This leads depending to the test level to the detection of faults and failures in the implemented mobile application. The knowledge about the findings are used to enhance the definition of the failure pattern classification for future development iterations. Furthermore, findings of unit tests, which are not part of the quality assurance method, support as well the enhancement of the failure pattern classification.

3.2 Exemplified Application of the Perspective-Based Inspection

To illustrate the inspection of the quality assurance method, two scenarios are presented in Tables 1 and 2 for quality assurance performed for a possible mobile application to be used for production process control. The requirements of this application are specified by screen mockups and free text. In this example, the defect classes and

failure classes are underlined to illustrate that these contents are variable according to the merge process and based on the failure pattern classification.

Table 1. Scenario "User Perspective".

Introduction	You are determined to use the mobile application after it has been fully developed, marketed, and installed. The end goal of the mobile application is to be useful to the consumer.
Instruction	Identify and mark parts of the requirements specification with an estimated high failure impact with "high risk" in the case of a missing interaction element.
Question	Are there parts of the specification with extraneous information?

A user, or an inspector taking the user's perspective, reads the introduction to adjust his mind to this perspective. Then he follows the instruction and answers the question shown in Table 1. The failure class *missing interaction element* is linked to a typical fault class, which will be used in the next scenario. The result in this example of the production process control application is that the inspector marks a designated status bar, which has to show critical warnings regarding the production process, with *high risk*. He does not find any extraneous information and quits the scenario.

Table 2. Scenario "Tester Perspective".

Introduction	You operate a mobile application under specified conditions, observing and recording the results and evaluating some aspects of the mobile application.
Instruction	Derive test cases considering the mark "high risk" due to a missing interaction element. Focus on failures that could be related to changing orientation or screen resolution.
Question	Are there parts of the specification with missing information?

A tester of the development project, respectively an inspector taking the tester's perspective, reads the introduction and follows the instructions of the scenario shown in Table 2. He considers the *high risk* mark for the derivation of test cases. Hence he knows that the marked part of the requirements specification is critical in case of the failure *missing interaction element* and that typical fault classes causing this failure are: a *changing orientation* or *screen resolution*. Consequently, he derives a test case, which will be focused on the typical failure pattern (shown in Table 3).

Table 3. Focused test case.

Precondition	Main screen viewed in horizontal orientation.
Action	Change orientation to vertical.
Postcondition	Status bar information completely visible.

While deriving the test case (Table 3), the inspector in the tester's role answers in parallel the question whether there is any *missing information*. In this example, it is the absence of information regarding how the application has to react to an orientation change. Several questions will arise depending to the experiences of the inspector: Is it allowed to cut interaction elements off? Is there a priority of displaying elements? Should vertical orientation be deactivated? Should the screen be scrollable?

The defects found in the requirements specification will be sent to the requirements engineers or designers and the test case will be performed as part of the system test by the tester, who thus has a focused test case to detect failures in the implemented mobile application.

3.3 Tool Prototype

In the previous example, the PBR scenarios were created manually. In order to enhance the efficiency of applying the method, the tool FOCUST (Focus and Control by Updated Scenarios and Tests) was initially developed. FOCUST represents a partial proof of concept by realizing the merge step of the method (see Sect. 3.1) in an automated way.

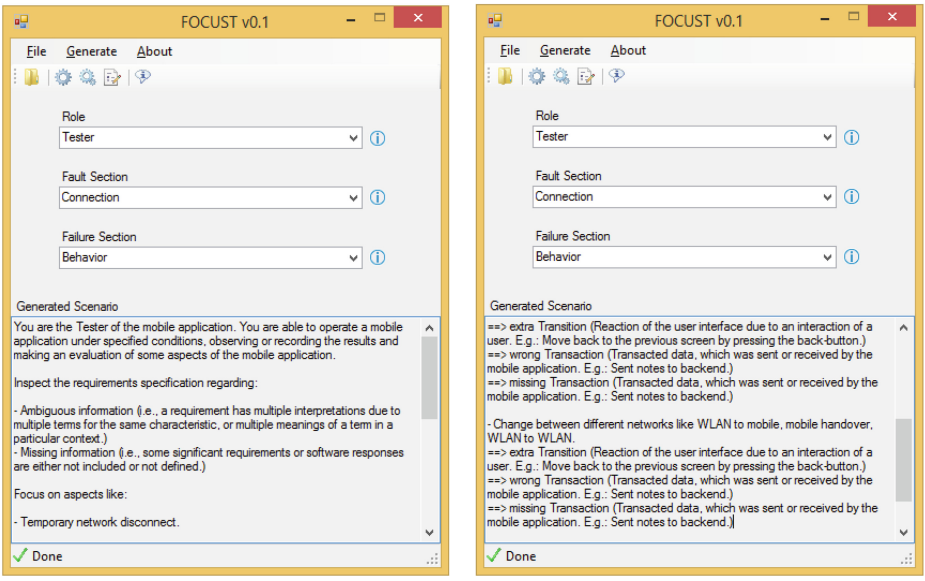

Fig. 4. Output example of the tool prototype for the tester's perspective.

FOCUST has an interface to the failure pattern classification including the relation to roles and defect classes. The user interface of FOCUST offers the selection of a specific role like *Tester* (see Fig. 4). Furthermore, the fault section (here: *Connection*) and the failure section (here: *Behavior*) can be selected. Based on that input, the

prototype generates basic PBR scenarios by listing the matching possibilities considering the given input. In future work, these basic scenarios will be linguistically enhanced according to Sect. 3.2.

3.4 Discussion

The development of the method and the partial implementation as a proof of concept revealed on the one side possible benefits due to the application of the method and led on the other side to open issues.

Benefits. The main advantage compared to common state-of-the-art approaches is that the developed quality assurance method focuses by different perspectives on typical failure patterns supposed to lead to a higher quality assurance effectiveness (see challenges in Sect. 1). Together with the PBR technique, the failure pattern classification [7] enables multiple benefits by being usable for inspection and testing activities through PBR scenarios. These PBR scenarios are efficient due to less overlapping of the findings of different inspectors [17]. A designer usually has another perspective and hence other findings than a tester or a user. Another benefit of those scenarios is the optimized work process during their performance. For the tester, this means that he derives test cases for those parts of the requirements specification that he is currently inspecting (i.e., checking for defects).

A defect in the requirements specification, such as ambiguous information, can cause a failure in the implemented mobile application. Hence, information about a found defect could be used to adjust the failure pattern classification as part of the fault classes and can be considered during the derivation of test cases. This can result in interaction between the testing and inspection activities and hence in a control loop (see Fig. 5), as both are related to the commonly used failure pattern classification.

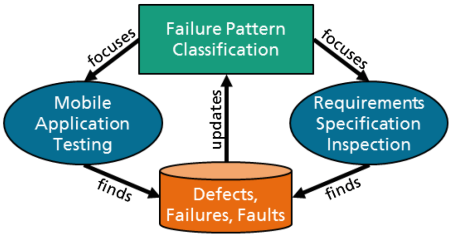

Fig. 5. Quality assurance control loop based on the information flow regarding the findings.

Open Issues. A crucial open issue is the missing basis to appraise the method's applicability within mobile application development projects with typical challenges like short iteration-cycles and a changing test basis. Insights for an evaluation could be reached by a first case study using the developed quality assurance method.

Before, several other issues needs to be solved. The scenarios cannot be generated completely due to a missing linguistic template. Furthermore, we need a determination of reasonable integration of the scenarios into the working process of the tester,

inspector, etc. That comprises, among others, the question “how to achieve an adequate cognitive load regarding the usage of the scenarios”.

4 Conclusions and Future Work

This work resulted in a FIT4Apps method to focus quality assurance of mobile applications and encourages research on several topics as future work.

4.1 Conclusion

The research question, “How to use the failure pattern classification to focus quality assurance?”, was answered by the development and exemplified partial application of the proposed quality assurance method, which uses the failure pattern classification presented in our previous contribution [7] (see Sect. 2.2). The combined testing and inspection method FIT4Apps based on the perspective-based reading technique enables the usage of the failure pattern classification. The resulting scenarios aim on focused defect detection by the inspection method and focused derivation of test cases considering typical failure patterns including their risks. The inspection may be applied to changing requirements specifications, working as a control loop, which could enable focused rework of quality assurance activities with reduced effort.

The developed method was partially implemented as a tool to support a quality assurance process through information processing, such as the generation of basic scenarios. These scenarios are actually based on previously defined input, like the derived failure pattern classification of the previous contribution [7].

Overall, we assume that the developed method could enable a higher quality assurance effectiveness, which can reduce the risk of failures remaining in mobile applications and hence their costly consequences.

4.2 Future Work

As future work, the integration of a defect causal analysis into the method’s control will be investigated considering the possibilities of defect prevention.

One of the most important further steps, besides the enhancement of the prototype, is the evaluation of the method to get evidence for its effectiveness and to obtain the possibility to compare it with other approaches (such as testing using defect taxonomies [18]). The primary metric will be the detection rates of defects and failures. This will be conducted first by a questionnaire consulting the target group and later by an experiment. One future research question will be about the indications, which exist that such our tailored quality assurance method is more efficient, stable and focused against a changing test basis.

Acknowledgment. The research described in this paper was conducted in the context of the Fraunhofer Project Center for Software and Systems Engineering at UFBA, a joint initiative of Fraunhofer Society and the Federal University of Bahia in Brazil, with support from the Bahia State Government.

References

1. Portio Research: Mobile applications futures 2013-2017, analysis and growth forecasts for the worldwide mobile applications market, pp. 22–28 (2013)
2. Aurum, A., Petersson, H., Wohlin, C.: State-of-the-art: software inspections after 25 years. Softw. Test. Verif. Reliab. **12**(3), 133–154 (2002)
3. Juristo, N., Moreno, A.M., Vegas, S.: Reviewing 25 years of testing technique experiments. Empir. Softw. Eng. **9**(1–2), 7–44 (2004)
4. Linz, T.: Testing in Scrum: A Guide for Software Quality Assurance in the Agile World (original title: Testen in Scrum-Projekten: Leitfaden für Softwarequalität in der agilen Welt, dpunkt.verlag), 1 edn. (2013)
5. Wasserman, A.: Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER), pp. 397–400. ACM, New York (2010)
6. Dantas, V.L.L., Marinho, F.G., da Costa, A.L., Andrade, R.M.C.: Testing requirements for mobile applications. In: 24th International Symposium on Computer and Information Sciences (ISCIS), pp. 555–560 (2009)
7. Holl, K., Elberzhager, F.: A mobile-specific failure classification and its usage to focus quality assurance. In: Euromicro Conference Series on Software Engineering and Advanced Applications (2014) (accepted)
8. Muccini, H., Di Francesco, A., Esposito, P.: Software testing of mobile applications: challenges and future research directions. In: 7th International Workshop on Automation of Software Test (AST), pp. 29–35 (2012)
9. Franke, D., Weise, C.: Providing a software quality framework for testing of mobile applications. In: IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST), pp. 431–434 (2011)
10. Makarand, T., Buenen, M.: World quality report key findings, executive summary. World Quality Report 2013–2014, Capgemini, Sogeti and HP, Fifth Edition, pp. 6–8 (2013)
11. Li, N., Li, Z., Sun, X.: Classification of software defect detected by black-box testing: an empirical study. In: Proceedings of Second World Congress on Software Engineering (WCSE), vol. 2, pp. 234–240. IEEE (2010)
12. Mauser, D., Klaus, A., Holl, K., Zhang, R.: GUI Failures of in-vehicle infotainment: analysis, classification, challenges and capabilities. Int. J. Adv. Softw. **6**, 142–154 (2013)
13. Chillarege, R.: Orthogonal defect classification. In: Lyu, M.R. (ed.) Handbook of Software Reliability Engineering, pp. 359–399. McGraw-Hill, New York (1996)
14. IEEE Standard Classification for Software Anomalies, IEEE Std., Rev. 1044-2009 (1994)
15. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report, version 2.3, Software Engineering Group (2007)
16. IEEE Recommended Practice for Software Requirements Specifications, IEEE Std., Rev. 830-1993 (1998)
17. Shull, F., Rus, I., Basili, V.: How perspective-based reading can improve requirements inspections. Computer **33**(7), 73–79 (2000). (IEEE Computer Society Press)
18. Felderer, M., Beer, A.: Using defect taxonomies for testing requirements. IEEE Softw. IEEE Computer Society Digital Library (2014)

An Exploratory Study on Risk Estimation in Risk-Based Testing Approaches

Michael Felderer(✉), Christian Haisjackl, Viktor Pekar, and Ruth Breu

Institute of Computer Science, University of Innsbruck, Innsbruck, Austria
{michael.felderer,christian.haisjackl,viktor.pekar,ruth.breu}@uibk.ac.at

Abstract. Risk estimation is a core activity in every risk-based testing process because it determines the significance of the risk values assigned to tests and therefore the quality of the overall risk-based testing process. In this paper we explore how risk estimation is performed in risk-based testing approaches. For this purpose, we classify 17 collected risk-based testing approaches according to predefined dimensions risk item type, factors, criteria, estimation technique, risk scale, estimation date, automation of measurement as well as tool support, and analyze the classification. Results from this classification reveal that a broad range of estimation variants is used but most approaches estimate risk for functional artifacts, consider probability and impact explicitly, use a quantitative scale and are based on manual measurement.

Keywords: Risk-based testing · Risk estimation · Risk assessment · Software risk management · Test management · Software testing

1 Introduction

Risk-based testing (RBT) is a type of software testing that explicitly considers risks of the software product as the guiding factor to solve decision problems in all phases of the test process, i.e., test planning, design, implementation, execution and evaluation. It is a pragmatic and well-known approach to address the problem of ever limited testing resources based on the intuitive idea to focus test activities on those scenarios that trigger the most critical situations for a software system [1]. Risk estimation is a core activity in every risk-based testing process because it determines the significance of the risk values assigned to tests and therefore the quality of the overall risk-based testing process.

A risk is the chance of injury, damage or loss and typically determined by the probability of its occurrence and its impact. As it is the chance of something happening that will have an impact on objectives [2], the standard risk formalization [3] is based on the two factors *probability* (P), determining the likelihood that a failure assigned to a risk occurs, and *impact* (I), determining the cost or severity of a failure if it occurs in operation. Consequently, estimating the risk exposure of a feature or component requires estimating both factors [4] either implicitly or explicitly.

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 32–43, 2015.
DOI: 10.1007/978-3-319-13251-8_3

Mathematically, the *risk exposure* R of an arbitrary asset a, i.e., something to which a party assigns value, is determined by the formula $R(a) = P(a) \circ I(a)$. The depicted operation $\circ$ represents a multiplication of two numbers or a cross product of two numbers or letters (and can principally be an arbitrary mathematical operation used to determine risk). In the context of testing, assets are arbitrary testable artifacts also called risk items. For instance, requirements, services, components or security risks are typical risk items for which risk exposure values R are estimated.

Within testing, a *risk item* is assigned to test cases which are typically associated with risk exposure values themselves derived from the risk items' risk exposure values. Risk exposure values are assigned to risk levels. A *risk level* [3] indicates the criticality of risk items and serves the purpose to compare risk items as well as to determine the use of resources. If explicitly considered, the factors P and I may be estimated directly or indirectly via intermediate *criteria* and metrics based on the Factor-Criteria-Metric model [5]. The metrics can be measured *manually* or *automatically*.

The use of risk estimation for software testing is considered in several studies, e.g., [4,6–8], but has so far not been investigated in a structured way. Kläs et al. [6] present an industrial case study on planning and adjusting quality assurance activities based on expert opinion as well as automatically collected metrics. Felderer et al. [7] present an approach to integrate manual and automatic measurement to estimate risks for testing purposes. Ramler and Felderer [4] discuss experiences from a study on risk probability estimation based on expert opinion. Finally, Haisjackl et al. [8] present a risk estimation tool for risk-based testing called RisCal.

In this paper we fill the gap of missing structured investigation of risk estimation for software testing by exploring how risk estimation is performed in risk-based testing approaches. For this purpose, we classify available risk-based testing approaches from literature according to predefined dimensions, and draw conclusions from it. The paper is an initial exploratory study with the primary aim to reveal the range of risk estimation approaches used for risk-based testing according to the classification dimensions risk item type, factors, criteria, estimation technique, risk scale, estimation date, automation of measurement, and tool support defined in Sect. 3.

The remainder of this paper is structured as follows. Section 2 presents the collected risk-based testing approaches. Section 3 defines classification dimensions for risk estimation, classifies the risk-based testing approaches according to them and discusses the results. Finally, Sect. 4 concludes and presents future work.

2 Risk-Based Testing Approaches

The overall purpose of risk-based testing approaches is to test in an efficient and effective way driven by risks. Every available risk-based testing (RBT) approach therefore integrates risk estimation into the testing process. In [9], the authors

present a risk assessment framework for testing purposes based on a systematic collection of 14 significant RBT approaches published in scientific literature. The collection was created on the basis of four recently published comprehensive journal articles on risk-based testing [10–13]. We take this collection of 14 RBT approaches as a basis and complement it by three relevant industrial RBT approaches reported in the literature [10,11] but not considered in [9], i.e., the approaches of Bach [14], Rosenberg [15], and van Veenendaal [16]. The resulting 17 risk-based testing approaches are shown in Table 1 ordered by the date of their first publication. Some approaches, i.e., Redmill, Stallbaum, Souza, as well as Felderer and Ramler are covered by more than one cited publication (see entries with identifiers 05, 06, 07 and 16 in Table 1).

3 Risk Estimation in RBT Approaches

In this section we present the classification dimensions for risk estimation in RBT approaches and their possible values (Sect. 3.1) as well as the concrete classification of the RBT approaches of Table 1 (Sect. 3.2).

3.1 Classification Dimensions for Risk Estimation

In the following we present the applied classification dimensions (also referred to as characteristics) for risk estimation in testing approaches, i.e., risk item type, factors, criteria, estimation technique, risk scale, estimation date, automation of measurement, and tool support. These characteristics and their possible values or items are based on the risk model of Sect. 1 as well as the risk assessment framework for testing purposes introduced in [9] and the taxonomy of risk-based testing presented in [30].

Risk Item Type. The risk item type determines the risk items, i.e., the elements to which risk exposure values and tests are assigned. Risk items can be of type *generic risk*, i.e., assets independent of a specific artifact like security risks, *test case*, i.e., directly test cases themselves as in regression testing scenarios, *runtime artifact* like deployed services, *functional artifact* like requirements or features, *architectural artifact* like component, or *development artifact* like source code file. The risk item type is determined by the test level. For instance, functional or architectural artifacts are often used for system testing, and generic risks for security testing.

Factors. Risk is determined by *probability* and *impact* resulting in a *risk exposure* value (see Sect. 1). This classification characteristic captures which of these factors are explicitly considered and estimated.

Criteria. Factors can be estimated indirectly via intermediate criteria like the Factor-Criteria-Metric model [5]. These criteria can be *business criteria* like monetary loss or *technical criteria* like complexity of components.

Table 1. Overview of identified risk-based testing approaches based on [9]

ID	Approach	Description
01	Bach [14]	The approach defines a heuristic risk estimation framework for testing. The underlying risk analysis can be either inside-out which begins with details of a situation (like concrete weaknesses or threats) and identifies risks, or outside-in which begins with a set of potential risks (defined in a list) and matches them to details of the situation. In addition, different ways to communicate risks and organize testing around them are defined, i.e., check lists, risk/task matrix, and component risk matrix
02	Amland [17]	The approach defines a process which consists of the steps (1) planning, (2) identification of risk indicators, (3) identification of cost of a fault, (4) identification of critical elements, (5) test execution as well as (6) estimation to complete. In addition, it is presented how the approach was carried out in a large project
03	Rosenberg [15]	The approach addresses risk-based testing of object oriented software. Risk is defined as the product of the severity of potential failure events and the probability of its occurrence. The failure probability is measured based on the metrics number of methods, weighted methods per class, coupling between objects, response of a class, depth of inheritance tree as well as number of children
04	Chen et al. [18]	The approach defines a specification-based regression test selection with risk analysis. Each test case is a path through an activity diagram (its elements represent requirements attributes) and has an assigned cost and severity probability. The test selection consists of the steps (1) assessment of the cost, (2) derivation of severity probability, and (3) calculation of risk exposure for each test case as well as (4) selection of safety tests. The risk exposure of test cases grouped to scenarios is summed up until one runs out of time and resources. The approach is evaluated by comparing it to manual regression testing
05	Redmill [19,20]	The approach reflects on the role of risk for testing in general and proposes two types of risk analysis, i.e., single-factor analysis based on impact or probability as well as two-factor analysis based on both factors
06	Stallbaum et al. [21,22]	The approach is model-based. Risk is measured on the basis of the Factor-Criteria-Metrics model and annotated to UML use case and activity diagrams from which test cases are derived

(*Continued*)

Table 1. (*Continued*)

ID	Approach	Description
07	Souza et al. [23,24]	The approach defines a risk-based test process including the activities (1) risk identification, (2) risk analysis, (3) test planning, (4) test design, (5) test execution, as well as (6) test evaluation and risk control. In addition, metrics to measure and control RBT activities are given. The approach is evaluated in a case study
08	Zimmermann et al. [25]	The approach is model-based and statistical using Markov chains to describe stimulation and usage profile. Test cases are then generated automatically taking the criticality of transitions into account. The approach focuses on safety-critical systems and its application is illustrated by examples
09	Kloos et al. [26]	The approach is model-based. It uses Fault Tree Analysis during the construction of test models represented as state machine, such that test cases can be derived, selected and prioritized according to the severity of the identified risks and the basic events that cause it. The focus of the approach are safety-critical systems and its application is illustrated by an example
10	Yoon and Choi [27]	The approach defines a test case prioritization strategy for sequencing test cases. Each test case is prioritized on the basis of the product of risk exposure value manually determined by domain experts and the correlation between test cases and risks determined by mutation analysis. The effectiveness is shown by comparing the number and severity of faults detected to the approach of Chen et al.
11	Zech [28]	The approach is model-based and derives a risk model from a system model and a vulnerability knowledge base. On this basis a misuse case model is derived and test code generated from this model is executed. The approach is intended to be applied for testing cloud systems
12	Bai et al. [10]	The approach addresses risk-based testing of service-based systems taking the service semantics which is expressed by an OWL ontology into account. For estimating probability and impact dependencies in the ontology are considered. The approach considers the continuous adjustment of software and test case measurement as well as of rules for test case selection, prioritization and service evaluation. The approach is evaluated by comparing its cost and efficiency to random testing

(*Continued*)

Table 1. (*Continued*)

ID	Approach	Description
13	Felderer et al. [7]	The approach defines a generic risk-based test process containing the steps (1) risk identification, (2) test planning, (3) risk analysis, (4) test design as well as (5) evaluation. Steps (2) and (3) can be executed in parallel. For this test process a risk assessment model based on the Factor-Criteria-Metrics model is defined. The metrics in this model can be determined automatically, semi-automatically or manually. The approach is illustrated by an example
14	van Veenendaal [16]	The approach called PRISMA (Practical Risk-Based Testing) provides a test process improvement approach based on product risk management. The standard PRISMA process consists of the activities (1) initiating, (2) planning, (3) kick-off meeting, (4) extended risk identification, (5) individual preparation, (6) processing individual scores, (7) consensus meeting, (8) define differentiated risk-based test approach, (9) monitoring and control, as well as (10) evaluation. Its risk model considers factors for impact of defects, mainly determined by business aspects, and their probability, mainly determined by technical aspects
15	Wendland et al. [1]	The approach is model-based. It formalizes requirements as integrated behavior trees and augments the integrated behavior tree with risk information. Then for each risk an appropriate test directive is identified, and finally both the risk-augmented integrated behavior tree and the test directive definition are passed into a test generator
16	Felderer and Ramler [13,29]	The approach defines a process to stepwise introducing risk-based testing into an established test process. On this basis four stages of risk-based test integration are defined, i.e., (1) initial risk-based testing including design and execution of test cases on the basis of a risk assessment, (2) risk-based test results evaluation, (3) risk-based test planning, as well as (4) optimization of risk-based testing. The approach is evaluated in a case study

(*Continued*)

Table 1. (*Continued*)

ID	Approach	Description
17	Ray and Mohapatra [12]	The approach defines a risk analysis procedure to guide testing. It is based on sequence diagrams and state machines. First one estimates the risk for various states of a component within a scenario and then, the risk for the whole scenario is estimated. The key data needed for risk assessment are complexity and severity. For estimating complexity inter-component state-dependence graphs are introduced. The severity for a component within a scenario is decided based on three hazard techniques: Functional Failure Analysis, Software Failure Mode and Effect Analysis and Software Fault Tree Analysis. The efficiency of the approach is evaluated compared to another risk analysis approach

Estimation Technique. The estimation technique determines how the risk exposure is actually estimated and can be *expert judgment* or *formal model* [31]. The essential difference between formal-model-based and expert-judgment-based effort estimation is the quantification step-that is, the final step that transforms the input into the risk estimate. Formal risk estimation models are based on a mechanical quantification step such as a formula or a test model. On the other hand, judgment-based estimation methods are based on a judgment-based quantification step-for example, what the expert believes is most risky. Judgment-based estimation processes range from pure gut feelings to structured, historical data and checklist-based estimation processes.

Risk Scale. The risk scale determines the level of risk measurement and can be *quantitative* or *qualitative*. Quantitative risk values are numeric and allow computations, qualitative risk values can only be sorted and compared. An often used qualitatively scale for risk levels is low, medium, high [1].

Estimation Date. The estimation date determines when the risk estimation takes place. This can be only *initially* as soon as risk items are available but before testing activities are performed or *iteratively* in each test cycle.

Automation of Measurement. Risk estimation can be supported by metrics which can be measured *manually* or *automatically*. The manual measurement is often supported by strict guidelines and the automatic measurement is often performed via static analysis tools.

Tool Support. Risk estimation can be supported by tools to perform it in a more efficient way. Software tools supporting estimation for testing may be

	RBT Approach																
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17
RiskItemType																	
Generic Risk	x								x	x	x			x	x		
Test Case				x	x												
Runtime Artifact		x			x			x				x					x
Functional Artifact	x	x			x	x	x						x	x	x	x	
Architectural Artifact	x		x		x								x	x		x	
Development Artifact			x		x											x	
Factors																	
Risk Exposure	x						x			x							
Probability		x	x	x	x	x		x	x		x	x	x	x	x	x	x
Impact		x	x	x	x	x		x	x		x	x	x	x	x	x	x
Criteria																	
Business Criteria		x			x	x	x						x	x		x	
Technical Criteria	x	x	x	x	x	x				x	x	x	x	x	x	x	x
Estimation Technique																	
Expert Judgment	x	x			x	x	x		x	x			x			x	
Formal Model		x	x	x		x		x			x	x	x	x	x		x
Risk Scale																	
Quantitative		x	x	x		x	x			x	x	x	x	x			x
Qualitative	x				x			x	x						x	x	
Estimation Date																	
Initial	x		x		x	x	x	x		x	x				x		
Iterative		x		x					x			x	x	x		x	x
Automation of Measurement																	
Manual Measurement	x	x		x	x	x	x	x	x	x		x	x	x	x	x	x
Automatic Measurement			x			x					x		x				x
Tool Support																	
Spreadsheets										x				x		x	
Specific Risk Estimation Tool						x	x								x		
Test Management Tool																	

Fig. 1. Risk estimation characteristics and classification of risk-based testing approaches of Table 1

spreadsheets, *specific risk estimation tools*, or *test management tools* either out of the box or customized.

3.2 Classification of RBT Approaches

Figure 1 shows the resulting classification of the collected 17 risk-based testing approaches according to the classification characteristics defined before. For each classified risk-based testing approach several values can be selected for each classification dimension, despite risk scale which is quantitative as soon as risk values are numeric and estimation date which is either initial or iterative.

With regard to the risk item type, the classification shows that more than 50 % of the RBT approaches (9 out of 17) assign risks to functional artifacts, i.e., requirements, use cases or system features. As most RBT approaches consider system-level testing, functional artifacts are the natural type of risk items supporting the early estimation of impact. To also consider probability on the basis of architectural or development artifacts for risk estimation, traceability between these artifacts and functional artifacts is required. Figure 2 shows the

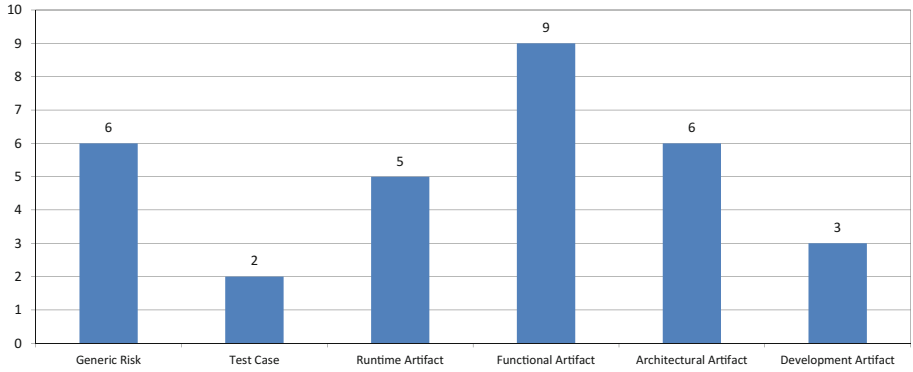

Fig. 2. Risk item types and their frequencies in the classified risk-based testing approaches

number of RBT approaches considering a specific risk item type. Besides functional artifacts, also generic risks (used in 6 approaches), architectural artifacts (used in 6 approaches) and runtime artifacts (used in 5 approaches) are quite common as they also play an important role for functional or non-functional system testing.

Most RBT approaches (14 out of 17) measure risks via probability and impact factors, and only two approaches measure risk exposure values directly without explicitly considering probability and impact factors. This indicates that measuring risk via probability and impact, which is sometimes also called cost, is widely used in established risk-based testing approaches. If probability and impact are considered, then technical criteria like complexity of components are often used to determine probability and business criteria like monetary loss are often used to determine impact. But it is also common to only consider technical or impact criteria. Most approaches (15 out of 17) are based on manual measurement. Whenever metrics are measured automatically, then technical criteria are considered for which automatic measurement is possible. For instance, the technical criterion complexity of a component can be measured automatically by the McCabe complexity [32] and the monetary loss can be estimated manually by a customer.

The estimation techniques expert judgment and formal models are almost equally applied. The used formal models are often based on fault models (e.g., in Zech [28]), the Factor-Criteria-Metrics approach (e.g., in Stallbaum et al. [21,22]), or on explicit test models (e.g., in Kloos et al. [26]).

An astonishing large number of RBT approaches (11 out of 17) uses a quantitative scale to measure risk. Several approaches, for instance Zech [28], Felderer [7] and van Veenendaal [16], combine quantitative and qualitative scales by mapping numeric risk exposure values to qualitative risk levels. These risk levels often combine qualitative scales for probability and impact each with values low, medium and high, which are then displayed in a 3×3 risk matrix.

Case studies or a survey on risk-based testing in industry could investigate whether the number of quantitative scales in industry is as high as in the analyzed publications.

About the same number of RBT approaches report initial (9 out of 17) and iterative estimation (8 out of 17). So not in all cases risks are periodically re-estimated to support decisions continuously but often are only used for initial decisions. Nevertheless, most RBT approaches for which only initial estimation is reported could also be estimated iteratively.

Only a few RBT approaches report on tool support. If tools are explicitly mentioned, then spreadsheets or specific risk estimation tools are used. None of the collected RBT approaches reports on the application of test management tools for risk estimation. Again, an empirical study in industry could reveal the actual situation in practice compared to the reported situation.

4 Conclusion

In this exploratory study we investigated how risk estimation is performed in risk-based testing approaches. For this purpose, we classified 17 relevant risk-based testing approaches from the literature according to the classification characteristics risk item type, factors, criteria, estimation technique, risk scale, estimation date, automation of measurement as well as tool support, and analyzed the results. The study reveals that a broad range of estimation variants is used, but most approaches estimate risk on the system level for functional artifacts, consider probability and impact explicitly, use a quantitative scale and are based on manual measurement. Expert judgment is almost as common as a formal estimation model, and so are initial and iterative measurement. Tool support for risk estimation is hardly reported.

These results of our study have several consequences for future research as well as the industrial application of risk-based testing. First, due to the high importance of risk-based testing for functional artifacts on the system level, research has to provide further traceability support to enable expressive formal models, to integrate technical criteria and to increase automation for more effective and efficient risk estimation. Second, our study shows that quantitative risk scales are used which would in principle enable predictive analytics which is very powerful but not commonly used in risk-based testing today. Therefore, research could provide suitable predictive analysis approaches, for instance based on bug prediction and system reliability models, and integrate them into risk-based testing approaches applied in practice. Third, additional tool support, which is according to our study rare at the moment, could support beforementioned traceability and predictive analytics but also estimation models as well as the automatic and iterative risk estimation. Finally, our results are based on reported academic and industrial RBT approaches. Case studies or a survey on risk-based testing in industry could investigate commonalities and differences to our results.

Acknowledgment. This research was partially funded by the research projects MOBSTECO (FWF P 26194-N15) and QE LaB - Living Models for Open Systems (FFG 822740).

References

1. Wendland, M.F., Kranz, M., Schieferdecker, I.: A systematic approach to risk-based testing using risk-annotated requirements models. In: ICSEA 2012, pp. 636–642 (2012)
2. Standards Australia/New Zealand: Risk Management AS/NZS 4360:2004 (2004)
3. ISTQB: Standard glossary of terms used in software testing, version 2.2. Technical report, ISTQB (2012)
4. Ramler, R., Felderer, M.: Experiences from an initial study on risk probability estimation based on expert opinion. In: IWSM-MENSURA 2013, pp. 93–97. IEEE (2013)
5. McCall, J., Richards, P., Walters, G.: Factors in software quality. Technical report, NTIS, vol. 1, 2 and 3 (1997)
6. Kläs, M., Elberzhager, F., Münch, J., Hartjes, K., von Graevemeyer, O.: Transparent combination of expert and measurement data for defect prediction: an industrial case study. In: ICSE 2010. ACM (2010)
7. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012)
8. Haisjackl, C., Felderer, M., Breu, R.: Riscal-a risk estimation tool for software engineering purposes. In: 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2013), pp. 292–299. IEEE (2013)
9. Felderer, M., Haisjackl, C., Pekar, V., Breu, R.: A risk assessment framework for software testing. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part II. LNCS, vol. 8803, pp. 292–308. Springer, Heidelberg (2014)
10. Bai, X., Kenett, R.S., Yu, W.: Risk assessment and adaptive group testing of semantic web services. Int. J. Softw. Eng. Knowl. Eng. **22**(05), 595–620 (2012)
11. Alam, M.M., Khan, A.I.: Risk-based testing techniques: a perspective study. Int. J. Comput. Appl. **65**(1), 42–49 (2013)
12. Ray, M., Mohapatra, D.P.: Risk analysis: a guiding force in the improvement of testing. IET Softw. **7**(1), 29–46 (2013)
13. Felderer, M., Ramler, R.: Integrating risk-based testing in industrial test processes. Softw. Qual. J. **22**(3), 543–575 (2014)
14. Bach, J.: Heuristic risk-based testing. Softw. Test. Qual. Eng. Mag. **11**, 99 (1999)
15. Rosenberg, L., Stapko, R., Gallo, A.: Risk-based object oriented testing. In: Proceedings of 13th International Software/Internet Quality Week-QW 2 (2000)
16. van Veenendaal, E.: Practical Risk-Based Testing - The PRISMA Approach. UTN Publishers (2012)
17. Amland, S.: Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. J. Syst. Softw. **53**(3), 287–295 (2000)
18. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research, p. 1. IBM Press (2002)
19. Redmill, F.: Exploring risk-based testing and its implications. Softw. Test. Verif. Reliab. **14**(1), 3–15 (2004)

20. Redmill, F.: Theory and practice of risk-based testing. Softw. Test. Verif. Reliab. **15**(1), 3–20 (2005)
21. Stallbaum, H., Metzger, A.: Employing requirements metrics for automating early risk assessment. In: Proceedings of MeReP07, Palma de Mallorca, Spain, pp. 1–12 (2007)
22. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test, pp. 67–70. ACM (2008)
23. Souza, E., Gusmao, C., Alves, K., Venancio, J., Melo, R.: Measurement and control for risk-based test cases and activities. In: 10th Latin American Test Workshop, pp. 1–6. IEEE (2009)
24. Souza, E., Gusmão, C., Venâncio, J.: Risk-based testing: a case study. In: 2010 Seventh International Conference on Information Technology: New Generations (ITNG), pp. 1032–1037. IEEE (2010)
25. Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T., et al.: Risk-based statistical testing: a refinement-based approach to the reliability analysis of safety-critical systems. In: EWDC 2009 (2009)
26. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safety-critical embedded systems driven by fault tree analysis. In: ICSTW 2011, pp. 26–33. IEEE (2011)
27. Yoon, H., Choi, B.: A test case prioritization based on degree of risk exposure and its empirical study. Int. J. Softw. Eng. Knowl. Eng. **21**(02), 191–209 (2011)
28. Zech, P.: Risk-based security testing in cloud computing environments. In: ICST 2011, pp. 411–414. IEEE (2011)
29. Felderer, M., Ramler, R.: Experiences and challenges of introducing risk-based testing in an industrial project. In: Bergsmann, J., Biffl, S., Winkler, D. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 10–29. Springer, Heidelberg (2013)
30. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. STTT **16**(5), 559–568 (2014)
31. Jorgensen, M., Boehm, B., Rifkin, S.: Software development effort estimation: formal models or expert judgment? IEEE Softw. **26**(2), 14–19 (2009)
32. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. **2**(4), 308–320 (1976)

Change Impact Analysis and Systems Testing

Improving Manual Change Impact Analysis with Tool Support: A Study in an Industrial Project

Thomas Wetzlmaier and Rudolf Ramler(✉)

Software Analytics and Evolution, Software Competence Center Hagenberg GmbH, Softwarepark 21, 4232 Hagenberg, Austria
{thomas.wetzlmaier, rudolf.ramler}@scch.at

Abstract. Change impact analysis is a challenging activity due to the usually huge number of dependencies that have to be considered. Nevertheless it is still often performed manually, relying on expert knowledge and intuition. The aim of this paper is to evaluate the practice of manual change impact analysis and to explore the benefits of tool support in the context of an industrial project. A study has been conducted with experienced developers estimating the changes necessary for implementing bug fixes and feature requests extracted from the project's history. The results of the manual change impact analysis showed a low estimation performance, which could be improved with tool support to achieve a higher number of hits at the expense of more false positives.

Keywords: Change impact analysis · Change prediction · Evolutionary coupling

1 Introduction

Change impact analysis is an important task in the incremental development, maintenance and evolution of software systems. It identifies the set of software entities that need to be changed when fixing a bug or when enhancing an existing system with a new feature. The information provided by change impact analysis supports a wide range of activities including planning and estimation, program comprehension, implementation, refactoring, as well as evaluation and testing.

In practice, analyzing the impact of changes can be a challenging task. Many software systems show characteristics such as a large size in terms of code base and implemented functionality, a high complexity due to feature interactions and technological dependencies, a rapid evolution due to a fast-paced agile development processes, and a substantial legacy in terms of existing code without sufficient documentation and regression tests.

In response, research has proposed numerous techniques and approaches for change impact analysis, which involve static program analysis, dynamic analysis and mining of software repositories [6, 9, 10]. Furthermore, many of these techniques and approaches have been implemented in tools providing automated support for change impact analysis, as their pure manual application is usually impractical or impossible due to the large amount of dependencies to be analyzed.

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 47–66, 2015.
DOI: 10.1007/978-3-319-13251-8_4

However, attempts to transfer change impact analysis techniques and accompanying tools to industry are often unsuccessful. For example, van de Laar [12] reports a case study conducted at Philips Healthcare MRI, where a tool implementing a state-of-the art change impact analysis approach has been rejected by the practitioners. Despite the tool's excellent prediction performance, the practitioners claimed that number of false positives and false negative produced by the tool was too high for industry use. As a consequence, many software projects still lack a systematic approach and subsequent automation support for change impact analysis. The pragmatic approach is usually based on the estimations of experienced developers or the discussions conducted in team meetings.

To explore the applicability and benefits of change impact analysis in practice, we conducted an experiment with experienced developers from an industrial project. The developers were asked to estimate the changes necessary for implementing different bug fixes or feature requests to the system. Change impact analysis was, first, performed manually – as practiced in the project – and, second, with additional support by a tool for exploring dependencies. The objective of the paper is to evaluate the practice of manual change impact analysis, to explore the benefits of automation support in an industrial context and, thereby, to contribute to the answer of the two general research questions: *(1) How well do experienced developers estimate changes?* and *(2) Can tool support improve the developers' estimates?*

The paper is structured as follows. Section 2 provides the necessary background and discusses the related work in the area of change impact analysis. Section 3 describes the organization of the study. The results of the study and its limitations are described in Sect. 4. The discussion of the results follows in Sect. 5. Finally, Sect. 6 presents our conclusions and plans for future work.

2 Background and Related Work

This section provides a brief introduction to change impact analysis, commonly used techniques and approaches, as well as suggested tool support. Furthermore, it presents related studies discussing manual versus automated change impact analysis.

2.1 Change Impact Analysis Process

Change impact analysis has been defined by Bohner and Arnold [1] as "the process of identifying the potential consequences of a change, or estimate what needs to be modified to accomplish a change". It is a key activity in the maintenance and evolution of software systems. The inputs for change impact analysis are change requests in the form of bug reports or requirement specifications together with the implementation of the existing system. The change requests are typically specified in natural language by the stakeholders, e.g., users, customers, or product managers. The output is a set of software entities that have to be changed in order to implement the bug fix or requirement. The general process for conducting a change impact contains three main steps that produce the following change sets:

1. *Initial Change Set*: Each change request is analyzed to estimate the corresponding *initial change set*. The change set includes the elements of the software system (e.g., modules, files, documents, tests) that need to be changed to fulfill the request.
2. *Extended Change Set*: Based on the initial change set, the impact set is determined by estimating the ripple effects of the changes. The estimated impact set contains the elements that are affected by the changes and, in consequence, also need to be changed. Typically, the result is an *extended change set* including the elements from the initially estimated change set as well as those from the estimated impact set.
3. *Actual Change Set*: Once the requested change has been implemented by modifying all directly or indirectly affected elements, the *actual change set* can be derived. The actual change set contains all elements that have actually been affected by the change.

In practice, these three steps are not clearly separated and there are many interactions as well as feedback loops between the different steps. For example, developers usually start to think about potential ripple effects concurrently to estimating the initial change set and, by exploring the potential impact of the changes they also reveal elements that should be part of the initial change set. The feedback from exploring the potential impact of a change may even lead to a discussion and adjustment of the change request originally specified by the stakeholders. Therefore we do not distinguish between change set and impact sets in this study. Instead we consider the union of these two sets when using the term change set.

2.2 Change Impact Analysis Techniques and Approaches

Since the work by Bohner and Arnold [1] in 1996, numerous further research works have proposed and explored a wide range of techniques and approaches for change impact analysis. The applied techniques range from (traditional) static and dynamic analysis of a program's structure and its execution to (more recent) approaches based on information retrieval and mining of software repositories – as well as any combination thereof. Furthermore, the research extended change impact analysis for program code to other types of software artefacts such as requirements, design or test cases and supports various application scenarios such as software comprehension, debugging, change propagation, or regression test selection.

Several attempts have been made to survey the existing research related to change impact analysis and to provide an overview by means of comparative studies. Li et al. [10] conducted a survey of change impact analysis techniques based on program source code. The survey includes 30 publications describing a total of 23 change impact analysis techniques from four perspectives based on software repository mining, execution information, traditional dependency analysis and coupling measurement. Lehnert developed a taxonomy for software change impact analysis [8] and conducted a comprehensive literature review [9] of 150 studies concerned with impact analysis of source code, architecture and requirements models, and miscellaneous artifacts (e.g., documentation, bug trackers, configuration files). He classified the studies according to the proposed taxonomy to provide a structured overview of the research field.

Cornelissen et al. [3] provided a literature survey on using dynamic analysis to support program comprehension, which also includes articles closely related to change impact analysis. They studied a total of 176 articles on program comprehension through dynamic analysis published between 1999 and 2008.

All surveys found a variety of techniques, which derive dependency data from the software change history stored in versioning systems such as CVS or Subversion (SVN) by mining these repositories. Such dependencies can be utilized for change impact analysis. Hence, the survey of Kagdi et al. [6] on mining software repositories considers change impact analysis as a typical application. The survey includes many studies that apply repository mining in context of change impact analysis and, thus, also provides a good overview of related works. This survey is particularly relevant in context of our work as we used an automated approach for change impact analysis based on repository mining.

2.3 Tool Support for Change Impact Analysis

Many of the different techniques and approaches are implemented by tools that provide automated or semi-automated support for program comprehension, feature localization as well as change impact analysis [10]. Many prominent examples of such tools have been released as research prototypes, which are described in detail in related publications.

JRipples is presented in the work by Buckner et al. [2] as a tool that assists programmers in performing impact analysis and change propagation when adding new functionality to an existing software system in incremental software development and evolution. The tool supports the analysis of Java programs. It parses the Java files to extract class dependencies, which are the basis for estimating the impact throughout incremental changes. JRipples has been implemented as a plug-in for the Eclipse and is available via jripples.sourceforge.net.

EvoLens has been developed and described by Ratzinger et al. [14]. The tool implements a visualization approach for explorations of evolution data across multiple dimensions. The graphical representation allows navigating through the hierarchical structure of a software system and along the time dimension by user-defined sliding time windows. The data basis is established by mining the version history of the analyzed system.

The tool *Hipikat* described by Cubranic and Murphy [4] is a recommender system for software development based on a broad collection of information about a project's history. This information is extracted from various sources including change tasks (i.e., bug reports and feature requests recorded in Bugzilla), source file versions (e.g., in a repository such as CVS), messages posted on developer forums (e.g., newsgroups and mailing lists), and other project documents (e.g., design descriptions). Change impact analysis is supported as one of the many different application scenarios of Hipicat [5].

Zimmermann et al. [20] presented the tool *ROSE* in context of their works on mining version histories to guide software changes by making suggestions in the form "programmers who changed these functions also changed …". The tool is implemented as an Eclipse plug-in. It extracts rules from a software system's version history and,

after an initial change made in the development environment, it predicts further locations to be changed.

The tool *HATARI* by Śliwerski et al. [17] takes this approach further by relating changes extracted from the version history (such as CVS) to a bug database (such as Bugzilla) to detect those locations where changes have been risky in the past. The risk is made visible for developers by annotating source code with color bars by providing the risk history of a particular location.

Chianti is an example where tool support for change impact analysis has been used in context of testing. The design and implementation of the tool has been described by Ren et al. [15]. The tool analyzes two versions of a Java software system and decomposes their difference into a set of atomic changes. Change impact is then reported in terms of affected (regression or unit) tests whose execution behavior may have been modified by the applied changes.

For improving the task of change impact analysis in software development, Pirklbauer et al. [13] suggested a change process accompanied by tool support. The tool is based on dependencies between artifacts extracted by mining CVS and SVN software repositories and analyzing check-ins as well as check-in comments. The resulting dependencies can be graphically visualized, interactively explored and adjusted to compute an estimated change and impact set (Fig. 1). We adopted the tool support for our study, in which it has been used to compute the dependencies of the studied project and to support the participants in performing change impact analysis tasks.

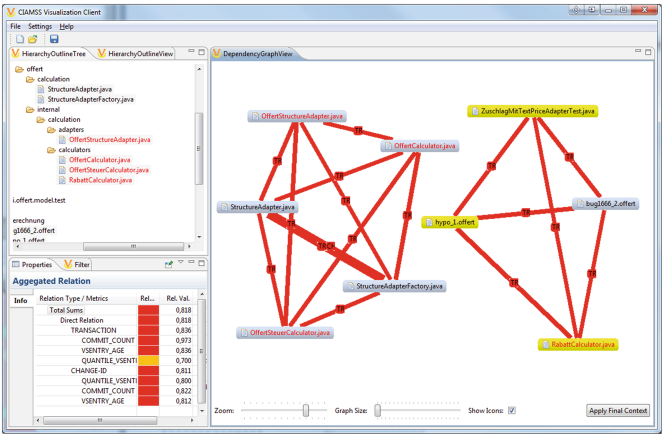

Fig. 1. Visualization of the dependency graph.

2.4 Manual Versus Automated Change Impact Analysis

In many software projects, change impact analysis is still a mainly manual activity conducted by developers and architects. Kilpinen [7] examined the applicability of proposed change impact analysis approaches and identified gaps in practical applications. To reflect the state of practice, she introduced the category of *Experiential Impact*

Analysis that refers to the widespread approaches based on tacit expert knowledge prevalent in informal team discussions and design or change review meetings.

However, the large size of current software systems and the often enormous number of dependencies between the involved entities limit the efficiency and effectiveness of manual approaches in change impact analysis [8]. The experiments and case studies conducted by Kilpinen [7] show that substantial benefits can be achieved when improving the support for pure manual change impact analysis in the development process. Nevertheless, there are only few studies of change impact analysis available that actually involve human experts, and results from automated approaches are rarely compared to results from manual approaches.

Lindvall and Sandahl [6] report on a long-term case study based on a development project of a telecom support system. The authors analyzed how well experienced, professional software developers are able to predict the necessary changes to implemented new requirements in the existing system. An approach called "requirements-driven impact analysis" was introduced to support developers in making predictions. The authors found that the correctness of the prediction is high although the under-prediction is worse than expected by well-regarded system developers. They also found a great improvement potential in using the requirements-driven change impact analysis approach, as the approach recommended additional classes to be taken into consideration by developers when making predictions. The paper also mentions that a large effort was necessary for the change impact analysis when using the tool-based approach.

Toth et al. [16] conducted an empirical comparison of four static impact analysis techniques (call information, static slice, SEA relation and co-changing files retrieved from SVN repositories) and dependencies which were estimated by programmers. The aim of the study was to investigate how well the different approaches supported program comprehension and change impact analysis. They found that the different automated approaches produced quite heterogeneous results and the intersection between the dependencies identified by the different approaches was rather small. Furthermore, when comparing the precision and recall rates of automated approaches to the collective programmer opinion they found a large deviation. They concluded "that no single algorithm can be relied upon and that the human opinion may depend on many different things" [16]. Finally, when allowing the developers to revise their estimated based on the results produced by the freely available tool JRipples they found that there was only a little improvement, but the programmers were generally open to accept new dependencies.

3 Study Organization

This sections describes the industry context in which our study took place, and the experiment design and procedures used to provide answers to the two initially stated research questions: "(1) How well do experienced developers estimate changes?" and "(2) Can tool support improve the developers' estimates?".

3.1 Analyzed Industry Project

The purpose of the software project that served as context for our study was the development of an ERP application for the building and construction industry. The project followed an agile process that fostered an incremental development over a timespan of more than eight years. The size of the development team varied throughout these years; most of the times it consisted of five to ten persons. In total, 15 different developers were involved in the project.

The software system, client/server application, has been based on Java, Eclipse RCP, JBoss, Hibernate, MySQL and various other technologies. The tools used by the developers included the Eclipse development environment, a SVN software repository, and Bugzilla for managing issues and change requests. These tools were tied together with Mylyn, providing task and application lifecycle management support integrated in Eclipse.

For our study, the project's version history of six consecutive years was made available. Table 1 shows the key characteristics of the project in this timespan as extracted from the SVN repository. An earlier report [19] describes the techniques applied for detecting dependencies and provides a detailed analysis of the change history found in the project's source code repository as well as the extracted dependency information.

Table 1. Data from 6 year project history.

Measure	Values
Number of Check-Ins	70,070
Number of Developers	15
Check-Ins/Developer [avg.]	4,671
Number of Transactions	8,134
Check-Ins/Transactions [avg.]	8.61
Number of Files	28,914
Check-Ins/File [avg.]	2.42
Number of Check-Ins with Change-ID [abs.]	21,905

3.2 Selection of Change Impact Analysis Cases

The decision was to use past changes recorded in the project's version history and to relate them to real bug fixes and feature requests for our study. We divided the project's history in two parts: (1) The first five years were used to populate the dependency database that built the data basis for the tool support. (2) The bug reports and feature requests selected for the experiment were drawn from the changes resolved in the sixth year.

In order to obtain a representative selection of changes for our experiment, we included cases with a variety of different characteristics (e.g., attribute values in the Bugzilla database) such as type of change (bug report, change request or feature request), severity and priority rating (high to low), number of check-ins required to complete the change (1–23). Table 2 shows the cases that were selected.

Table 2. Cases selected for the experiment.

Case No.	Change Type	Priority	Changed Files	Added Files	Check-Ins	Assigned to
1	change	low	21	37	3	A
2	change	high	6	6	4	C
3	change	high	3	3	1	C
4	feature	high	4	3	2	C
5	feature	high	27	39	2	C
6	feature	normal	4	4	1	B
7	change	high	4	1	1	C
8	change	normal	1	0	1	B
9	bug	normal	1	0	1	A
10	bug	normal	58	0	1	B
11	bug	high	7	0	1	B
12	bug	high	1	0	1	B
13	bug	high	3	2	4	A
14	change	low	3	2	1	C
15	bug	high	3	0	2	A
16	feature	high	5	119	23	A
17	bug	high	1	1	1	B
18	feature	high	3	8	7	C
19	change	high	4	1	1	C
20	feature	high	2	8	3	B
21	bug	normal	1	1	1	C
22	change	high	8	10	1	C
23	change	low	1	0	1	A
24	feature	low	2	36	4	A
25	feature	high	1	12	3	B
26	feature	high	12	29	2	A
27	feature	high	2	11	4	B
		min	1	0	1	
		max	58	119	23	
		median	3	3	1	
		average	7,0	12,3	2,9	
		std.dev	11,9	24,5	4,3	

Three different *types of change* were distinguished in the Bugzilla database: “bug report”, “change request” and “feature request”. Bug reports describe failures and the related change set usually involves only a few modified files. Change requests affect the existing functionality and, thus, the change set involves existing as well as newly added files. The number of added files is generally low and typically also lower than the number of changed files. Feature requests concern new functionality that has to be added to the system. This is also reflected in the number of newly added files, which is generally high and typically much higher than the number of changed files.

3.3 Participants and Tool Support

The study participants were three developers from the project team who had participated in the team for several years. Each of them had extensive knowledge about the system and its implementation. Thus, when assigning the cases to be estimated to participants, we made sure that none of the participants was assigned a case that he had actually resolved in the past.

In a preparation meeting, we explained the details and purpose of the experiment to the participants. Furthermore, they received a brief hands-on tutorial introducing the provided tool support for change impact analysis as described in [1]. When conducting the experiment, the participants used the provided tool as well as their personal development environment for examining the source code for potentially necessary changes.

3.4 Experiment Setup and Execution

Each participant received a share of the selected cases (a mix of bug reports, change and feature requests) together with the system's source code in the state before the corresponding changes had been made. Each case included the full description and any optional attachments from the Bugzilla database.

The goal of the estimate was defined as follows: "Identify the files in the code base that have to be changed in order to implement the specified bug fix, change or feature request." For each case the participants were asked to conduct series of estimates producing five result sets. Figure 2 provides an overview of the sequence in which the estimates were made and how they depend on each other.

1. **ICS-Adhoc** (*Ad hoc Initial Change Set*): The first estimate that had to be made was an ad hoc estimate based on the description of the requested change and the system's file structure. The participants were asked to produce the estimate spending only a few minutes on the case. The time limit reflects the typical situation of giving a quick advice when asked about a change.
2. **ICS-Informed** (*Informed Initial Change Set*): For the second estimate of the same case, the participants were asked to take as much time as needed for carefully analyzing the necessary changes in the code base. The participants were able to consult the documentation of the system and to use their development environment to investigate implementation details in order to identify relevant files and dependencies. This step reflects the typical change analysis approach in the project.
3. **ECS-Informed** (*Extended Change Set derived from the Informed Initial Change Set*): The tool support was used to extend the initial change set produced in the informed estimate. The tool computed the potential impact of the initial change set and suggested additional files that may also need to be changed. All changes suggested by the tool were included in the change set produced in this step to reflect the (worst) case of overestimation.
4. **ECS-Revised** (*Revised Extended Change Set derived from the Informed Initial Change Set*): The participants were asked to revise the previously extended change set that included all changes suggested by the tool. The participants were encouraged

to "play" with the tool and to fine-tune the computed results. In the end, the revision could be anything from keeping the tool's suggestions to removing them entirely, as well as reconsidering the initial change set produced by the informed estimated under the light of the suggestions made by the tool. This step reflects the suggested use of the tool support for change impact analysis.

5. **ECS-Adhoc** (*Ad hoc Extended Change Set*): Finally, the tool support was also used to extend the initial change set produced in the ad hoc estimate. Again, all changes suggested by the tool were included in the change set. The obtained result reflects the application of the tool by an uninformed user seeking a quick "second opinion".

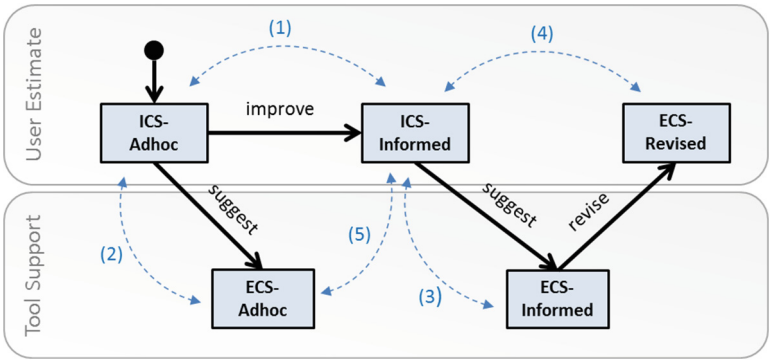

Fig. 2. Estimates made in the experiment (black arrows = sequence of estimates, blue dashed arrows = comparisons for evaluation) (Color figure online).

3.5 Evaluation Procedure and Measures

Each of the five change sets is compared to the *actual change set* (ACS). The actual change set is the set of actually changed files in the SVN repository associated with the resolved bug report, change or feature request in the Bugzilla database. For the comparison we use the measures *Recall* and *Precision*, which are frequently used in information retrieval [16]. Recall is the fraction of correctly estimated changes in the actual change set. It is computed for an estimated change set *xCS* and the actual change set ACS as

$$\text{Recall}(\text{xCS}) = |\text{xCS} \cap \text{ACS}|/|\text{ACS}| \quad (1)$$

Precision is the fraction of correctly estimated changes in the estimated change set. It is computed for an estimated change set *xCS* and the actual change set ACS as

$$\text{Precision}(\text{xCS}) = |\text{xCS} \cap \text{ACS}|/|\text{xCS}| \quad (2)$$

These two measures cannot be evaluated separately as an increase of Recall usually leads to a decrease in Precision and vice versa. In most cases the preference of one measure over the other depends on the users' priorities and risk models. In our

evaluation, we value Recall higher than Precision since we assume that false positive (superfluous) estimates can be easily identified and discarded when implementing the change, while false negative (missing) estimates will most likely remain unnoticed also in later activities.

The Precision and Recall values of the individual estimates are aggregated over all cases for each of the five change set types to support the comparisons shown as blue dashed lines in Fig. 2. Based on these comparisons, the following questions will be discussed:

1. *How do developer estimates improve when they have enough time for analyzing the changes?* (ICS-Adhoc vs. ICS-Informed).
2. *How do the suggestions made by the tool improve the ad hoc estimates of the developers?* (ICS-Adhoc vs. ECS-Adhoc).
3. *How do the suggestions made by the tool improve the informed estimates of the developers?* (ICS-Informed vs. ECS-Informed).
4. *How do developers revise and improve their informed estimates with tool support?* (ICS-Informed vs. ECS-Revised).
5. *Is it better to invest time and effort in manually analyzing changes or to rely on tool suggestions?* (ECS-Adhoc vs. ICS-Informed).

4 Results

Table 3 shows the evaluated estimates for the 27 selected and processed cases. Each case is identified by the case number given in the first column (*Case No.*) of the table. The second column (*ACS size*) reports the size of the actual change set, i.e., the number of actually changed files in the code base. The actual change set served as reference for evaluating the participants' estimates.

For each case the participants provided estimates for the five different change sets described in Sect. 3.4 (columns *ICS-Adhoc*, *ESC-Adhoc*, *ICS-Informed*, *ECS-Informed* and *ECS-Revised*). Exceptions (marked as gray cells in the table) are the cases #1 for which only the ad hoc estimate ICS-adhoc and the derived extension were provided, and the cases #25, #26 and #27 for which only the informed estimate ICS-Informed and the subsequent estimates were provided. In these cases the participants simply forgot to record the most recent list of identified files before submitting the estimates.

The estimates for each change set are described by the values *Size*, *Recall* and *Precision* (see corresponding columns in the table). Size is the number of files in the change set. The measures Recall and Precision were computed as explained in Sect. 3.5. A considerable number of change sets do not contain any correctly estimated change. Thus, the Recall (and the Precision) of these estimates is 0 %. To clearly distinguish these estimates from those estimates that had at least one hit, the estimates with a Recall > 0 (hit) are printed in bold font with a green background.

For each column, the measures *min*, *max*, *median*, *average* and *standard deviation* have been computed. Furthermore, *recall > 0* shows the percentage of estimates with at least on correctly identified change for each of the five different change sets. The statistical measures are heavily impacted by the large number of cases for which the

Table 3. Overview of the estimation results (highlighted cells indicate a Recall > 0, gray cells indicate missing data).

Case	ACS	ICS-Adhoc			ECS-Adhoc			ICS-Informed			ECS-Informed			ECS-Revised		
No.	Size	Size	Rec%	Pre%	Size	Rec%	Pre%	Size	Rec%	Pre%	Size	Rec%	Pre%	Size	Rec%	Pre%
1	21	2	**5**	**50**	65	**71**	**23**									
2	6	3	**17**	**33**	60	**83**	**8**	2	**33**	**100**	76	**100**	**8**	76	**100**	**8**
3	3	3	**33**	**33**	272	**100**	**1**	3	**33**	**33**	272	**100**	**1**	3	**33**	**33**
4	4	3	**25**	**33**	166	**100**	**2**	4	**25**	**25**	193	**100**	**2**	3	**25**	**33**
5	27	6	**4**	**17**	143	**22**	**4**	2	**7**	**100**	108	**30**	**7**	5	**11**	**60**
6	4	1	**25**	**100**	157	**25**	**1**	1	**25**	**100**	157	**25**	**1**	1	**25**	**100**
7	4	1	**25**	**100**	43	**75**	**7**	1	**25**	**100**	43	**75**	**7**	34	**75**	**9**
8	1	2	**100**	**50**	47	**100**	**2**	1	**100**	**100**	36	**100**	**3**	1	**100**	**100**
9	1	1	**100**	**100**	74	**100**	**1**	1	**100**	**100**	74	**100**	**1**	51	**100**	**2**
10	58	28	**47**	**96**	39	**50**	**74**	28	**47**	**96**	39	**50**	**74**	39	**50**	**74**
11	7	3	**0**	**0**	85	**100**	**8**	5	**29**	**40**	149	**100**	**5**	145	**100**	**5**
12	1	2	**0**	**0**	5	**0**	**0**	4	**100**	**25**	38	**100**	**3**	38	**100**	**3**
13	3	2	**0**	**0**	9	**0**	**0**	2	**67**	**100**	9	**100**	**33**	4	**67**	**50**
14	3	1	**0**	**0**	2	**0**	**0**	1	**33**	**100**	2	**33**	**50**	1	**33**	**100**
15	3	1	**0**	**0**	25	**0**	**0**	3	**0**	**0**	51	**33**	**2**	19	**33**	**5**
16	5	1	**20**	**100**	103	**100**	**5**	1	**0**	**0**	45	**80**	**9**	44	**80**	**9**
17	1	6	**0**	**0**	6	**0**	**0**	2	**0**	**0**	29	**100**	**3**	2	**0**	**0**
18	3	3	**0**	**0**	98	**33**	**1**	2	**0**	**0**	52	**0**	**0**	5	**0**	**0**
19	4	1	**0**	**0**	1	**0**	**0**	1	**0**	**0**	1	**0**	**0**	1	**0**	**0**
20	2	2	**0**	**0**	5	**0**	**0**	5	**0**	**0**	42	**0**	**0**	28	**0**	**0**
21	1	1	**0**	**0**	34	**0**	**0**	1	**0**	**0**	34	**0**	**0**	10	**0**	**0**
22	8	2	**0**	**0**	2	**0**	**0**	4	**0**	**0**	4	**0**	**0**	4	**0**	**0**
23	1	1	**0**	**0**	11	**0**	**0**	3	**0**	**0**	20	**0**	**0**	19	**0**	**0**
24	2	4	**0**	**0**	51	**0**	**0**	2	**0**	**0**	37	**0**	**0**	28	**0**	**0**
25	1							5	**0**	**0**	5	**0**	**0**	5	**0**	**0**
26	12							4	**0**	**0**	20	**0**	**0**	10	**0**	**0**
27	2							1	**0**	**0**	1	**0**	**0**	0	**0**	**0**
min	1	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0
max	58	28	100	100	272	100	74	28	100	100	272	100	74	145	100	100
median	3	2	0	0	45	23,6	0,8	2	3,7	12,5	38,5	33,3	1,7	7,5	25,0	3,7
average	7,0	3,3	16,7	29,7	62,6	40,0	5,8	3,4	24,0	39,2	59,1	47,2	8,1	22,2	35,9	22,8
std.dev	11,9	5,5	28,9	40,0	67,1	43,7	15,5	5,2	33,3	46,2	66,2	44,9	17,6	31,8	40,1	35,0
recall > 0			46%			54%			50%			62%			58%	

participants were not able to provide correct estimates (Recall = Precision = 0.00). To gain a better insight about the more successful cases with Recall and Precision values > 0, we calculated the statistics for these cases separately. The results are depicted for Recall in Fig. 3 and for Precision in Fig. 4.

5 Discussion

This section discusses the estimation results in terms of the five comparisons suggested in Sect. 3.5 and highlights the main observations. Furthermore, it lists the limitations of the study and the involved threats to validity.

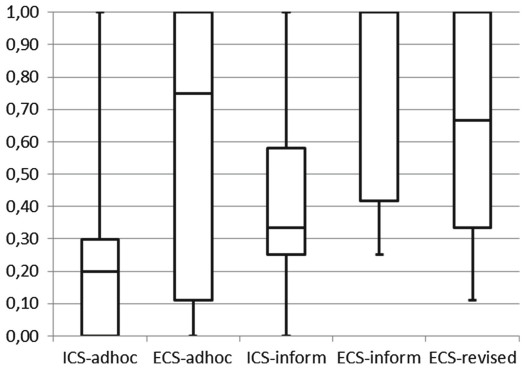

Fig. 3. Recall for cases with at least one correct estimate.

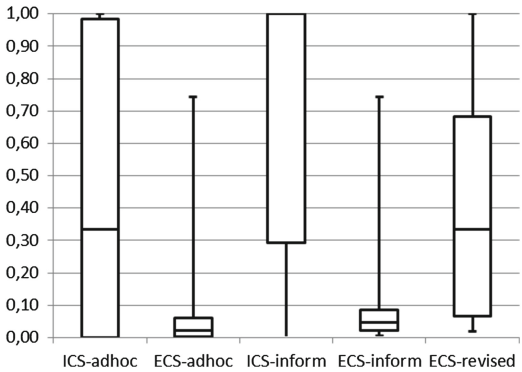

Fig. 4. Precision for cases with at least one correct estimate.

5.1 Comparisons

An overview of the comparisons discussed in the following is provided in Table 4. The table shows how the compared estimates (columns *left* and *right*) have been produced, i.e., by human estimation based on manual change analysis (*H*) or with the automated support of the tool (*T*). Furthermore, the percentage values in the columns *Recall delta* and *Precision delta* show the delta between the right and the left hand side of the comparison. For example, by extending the ICS-Adhoc change set to ICS-Informed the median Recall increases by +13 % and the median Precision by +67 %.

(1) ICS-Adhoc vs. ICS-Informed: How do developer estimates improve when they have enough time for analyzing the changes? We found that **a thorough analysis is able to** increase **Recall and Precisions at the same time, so it is most valuable for improving estimates of the initially proposed change set.** This finding is based on following observations:

Table 4. Overview of comparisons.

Left	Right	Comparison		Recall delta	Precision delta
H	H	(1)	ICS-Adhoc vs. ICS-Informed	13 %	67 %
H	T	(2)	ICS-Adhoc vs. ECS-Adhoc	55 %	−31 %
H	T	(3)	ICS-Informed vs. ECS-Informed	67 %	−95 %
H	H + T	(4)	ICS-Informed vs. ECS-Revised	334 %	−67 %
T	H	(5)	ECS-Adhoc vs. ICS-Informed	−42 %	98 %

- The number of cases in which the developers were not able to identify any of the actually changed files was quite high. With ad hoc estimates they achieved a Recall > 0 only in 46 % of the cases – 11 out of 24.
- The developers were able to improve through a detailed analysis and found four more such cases (#11 to #14) with at least one changed artefact, thus, increasing the share of cases with a Recall > 0 to 50 %. In one case (#16), however, the user removed the one correctly estimated change and selected a not affected artefact.
- The size of both, the ad hoc and the informed estimated change sets is almost the same. The average size increased from 3.3 files to 3.4 files. In some cases a few files were added, in others a few files were removed, or the set was kept the same.
- The average Recall rose from 17 % to 24 %, and the average Precision from 30 % to 39 %. This is the only comparison where an increase in both measures, Recall and Precision, could be observed.

(2) ICS-Adhoc vs. ECS-Adhoc: How do the suggestions made by the tool improve the ad hoc estimates of the developers? We found that **the automation support considerably increases the size of the estimated change set, which results in a higher Recall but a sharp decrease of the Precision**. This finding is based on following observations:

- The tool extended the average size of the change set estimated by the developers from an average of 3.3 files to an average of 62.6 files when all tool suggestions were added. The maximum size of the extended set is 272 files even though the initial size was 3 files. Yet for six cases the tool provided no or only a small number of suggestions (0–3 additional files).
- In two cases (#11, #18) in which the users were unsuccessful in providing a correct initial estimate (Recall = 0), the tool support was able to improve the initial estimate and produced an extended change set that included actually affected files. In both cases the size of the estimated change set had been tremendously increased, from 3 to 85 files and from 3 to 98 files respectively.
- The average Recall more than doubled from 17 % to 40 % but the average Precision dropped from 30 % to 6 %. The situation remains the same when ignoring all unsuccessful estimates (Recall = 0).

(3) ICS-Informed vs. ECS-Informed: How do the suggestions made by the tool improve the informed estimates of the developers? We again found that **the automation**

support considerably increases the size of the estimated change set, which results in a moderately higher Recall but a sharp decrease of the Precision. **The tool support had the same effect as when applied to the ad hoc estimates.** These findings are based on following observations:

- All observations and, thus, also our findings are very similar to those made in the previous comparison (*ICS-Adhoc vs. ECS-Adhoc)*, even though that basis on which the tool had been applied had been considerably improved (see *ICS-Adhoc vs. ICS-Informed*).
- The tool extended the average size of the change set estimated by the developers from an average of 3.4 files to an average of 59.1 files when all tool suggestions were added. The maximum was again the case with 272 files.
- Three previously unsuccessful predictions (#15to #17) were improved to a Recall > 0 by the tool support. Again, each improvement seems to be the result of a massive increase in the size of the change set.
- The average Recall also approximately doubled, in this instance from 24 % to 47 % and the average Precision dropped again notably, in this instance to about one fifth from 39 % to 8 %.

(4) ICS-Informed vs. ECS-Revised: How do developers revise and improve their informed estimates with tool support? This comparison reflects a complete change impact analysis process including the use of tool support. We found that **the interactive application of the tool resulted in a slight improvement of the Recall that is weakened by a loss of Precision**. **In a setting where Recall is preferred over Precision, these results can still be considered moderately positive. Three application scenarios can be distinguished, in which the users mainly (a) accepted, (b) revised or (c) discarded the suggestions of the tool.** These findings are based on following observations:

- The average size of the change set increased from 3.4 files to 22.2 files, with a maximum of 145 files. This increase is due to the interactive revision of the suggestions made by the tool (see *ICS-Informed vs. ECS-Informed*).
- In two cases (#15, #16) the initially unsuccessful estimate of the developer could be turned into an estimate with a Recall > 0 based on the suggestions provided by the tool. In another case (#17), however, the correct suggestion of the tool was discarded in the revision process.
- The tool suggestions based on an informed estimate and combined with the revision by the developers led to a slight decrease in the overall prediction performance, which has to be attributed to the loss of Precision (39 %–23 %) that could not be compensated by the increase of the Recall (24 %–36 %).
- When the developers interactively reworked the tool's suggestions, they always improved the Precision. Thereby, in several instances, the Recall was decreased when compared to the full set of suggestions produced by the tool. The revision of the tools suggestions (ECS-Informed) is therefore essential for improving the estimates.
- For five cases the tool was not able to provide a suggestion. In the other 21 cases the tool extended the developers' initial estimate; 7–269 suggested files were added.

In 6 cases the developers decided to keep the added suggestions almost unchanged (0–6 % of the suggestions were removed), in 7 cases the developers discarded the suggestions almost entirely (0–6 % were kept), and in 8 cases the suggestions were revised (on average 49 % were removed).

(5) ECS-Adhoc vs. ICS-Informed: Is it better to invest time and effort in manually analyzing changes or to rely on tool suggestions? We found that **a manual improvement of the initial estimates achieves better results than solely relying on the tool's suggestions**. This finding is based on following observations:

- The comparison of the Recall values (40 % vs. 24 %) suggests a potential advantage of the tool, which is due to the massive number of additional suggestions. The tool support led to more than 1,400 additional files, while the developers' informed estimates led to only 1 additional file. The consequences are also clearly visible in the different Precision values (6 % with the added tool suggestions vs. 39 % for informed manual estimates).
- However, the effects of the informed estimates seem to run even deeper. While the tool support helped to find actually affected files in two cases for which the initial ad hoc estimates of the developers were initially unsuccessful (Recall = 0), their informed estimates improved four such cases. One can therefore assume that in the process of making a more informed estimation the developers also gain additional insights and knowledge about the system and the relevant dependencies.

A. General Observations and Lessons Learned

In the course of our study we made additional observations that led to the following insights.

- *A high number of estimates are "unsuccessful", i.e., do not identify any of the actually changed files:* In half of the cases, the developers were not able to produce an estimation that contained even a single correctly estimated file. This generally high number of "unsuccessful" estimates existed in both of the scenarios, regardless whether estimates were made ad hoc or informed. This observation seems in consent with the observation mentioned by Lindvall and Sandahl, where the developers were "surprised by the large discrepancies between the prediction and the actual outcome" [11].
- *The main benefit of the automation support is the large number of additional suggestions:* The tool generally demonstrated the typical effect of "conservatively" increasing the recall at the cost of decreasing the precision. This effect was observable in an equal manner in the ad hoc and in the informed estimation scenario. In the latter one, however, it provided the benefit that the best estimates of the developers, for which they had "as much time as needed for carefully analyzing the necessary changes in the code base", could still be further improved.
- *Automation support provides a starting point for further manual change impact analysis:* The tool cannot be productively applied without revising the generated results. This revision can be done as part of the change impact analysis process or it can be left to the developer actually implementing the change. In either case the tool's suggestions are mainly a starting point guiding the user to the potentially

relevant dependencies. The actual investigation of these dependencies always incurs further manual work. The time and effort necessary for producing accurate estimates can therefore be quite high. In our study, the participants required four to six hours each for the assigned cases.

5.2 Limitations and Threats to Validity

The industrial context of our study and its limitations have to be considered when interpreting the results and drawing conclusions.

The tool used in the study to support the developers in making and revising estimates was at the level of a stabilized prototype. The dependency database was constructed and evaluated before the experiment to make sure the data basis was correct and complete. Nevertheless, the included dependencies were entirely based on the project's change history and may not capture the entire set of existing dependencies [18]. For example, the repository mining approach implemented by the tool did not analyze the static dependencies in the source code. In the experiment the participants used their development environment to analyze relevant dependencies of this type.

Furthermore, for accessing and analyzing the dependency data stored in the tool's database, participants had to use the tool's graphical interface. The effort for understanding and using the tool as well as usability issues of the tool support such as its low performance may have had an impact on the analysis and the estimation results.

The study was conducted ex-post, based on historical project data. Developers were assigned cases to be estimated based on bug reports and feature requests from the project's history. We made sure that none of the developers was assigned a case that he had actually resolved in the past. This strategy avoided that developers remembered their actual changes instead of estimating the changes. However, it introduced a bias since developers may have been assigned cases outside of their main area of knowledge.

The ex-post approach of the study required developers to "travel back in time" and to analyze the change impact in context of the specific historic state of the system. Hence, the estimation performance of the developer may have been influenced by the additional effort required to recollect relevant details and dependencies "at that time".

The study has been conducted in context of a specific project. The selected cases as well as the involved tasks can be considered representative for this project. The involved developers had been involved in the project for several years and contained a detailed knowledge about the software system and excellent technical skills. However, the findings of the study may not be generalizable beyond projects and organizations with similar characteristics.

6 Conclusions and Future Work

Despite the usually huge number of dependencies in modern software systems, change impact analysis is still a mainly manual task relying on expert knowledge and intuition. In this paper we reported the results of an experiment that involved experienced

developers from an industrial project estimating the changes necessary for implementing bug fixes and feature requests extracted from the project's history. The developers were asked to provide a series of change estimates ranging from pure manual estimates conducted in an ad hoc fashion to estimates based on a thorough tool-supported analysis of the dependencies.

The results related to the first research question *"(1) How well do experienced developers estimate changes?"* can be summarized as follows. First of all, the results show a surprising low prediction performance by the developers. We found a high number of cases – 46 % of all estimates – in which the developers were not able to identify at least one of the files that were changed when the bug report or feature request had actually been resolved. The results appear less disastrous when the "category of unsuccessful estimates" is ignored; for those cases where the developers "had a clue" when making estimations the best average recall values reached 62 %, best average precision 78 %. The positive aspect we found is that prediction results improved with an increasing investment of time and effort in estimating the changes.

Concerning the second research question *"(2) Can tool support improve the developers' estimates?"* the results indicate a general overestimation when the suggestions produced with the tool were added to the developers' estimates. The average size of the estimated change sets rose from initially 3 to about 60 files to be considered; in several cases the change set was extended to far more than 100 files. The observable effect was an increase of the recall accompanied by a sharp decrease of the precision. The benefit of the tool support is therefore limited. Nevertheless, developers favoring a "conservative" estimation scenario may still consider these results as an acceptable improvement [11]. Thus, when the developers had the chance to revise the extended change set, we observed several cases in which they decided to keep most of the added suggestions.

By studying one specific project, we were able to present only a glimpse in the situation prevalent in practice. The replication and extension of this study is strongly recommended and part of our plans for future work. Moreover, the tool support used in this study may have introduced limitations that are specific for the approach implemented in this particular tool. In our future work we intend to apply additional tools to extend the range of automation support.

Acknowledgements. This work has been supported by the competence centers program COMET of the Austrian Research Promotion Agency (FFG). Furthermore, the authors would like to thank the developers involved in the analyzed industry project for participating in the study.

References

1. Bohner, S.A., Arnold, R.S.: Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos (1996)
2. Buckner, J., Buchta, J., Petrenko, M., Rajlich, V.: JRipples: a tool for program comprehension during incremental change. In: Proceedings of the 13th International Workshop on Program Comprehension (IWPC '05), pp. 149–152. IEEE Computer Society (2005)

3. Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. IEEE Trans. Softw. Eng. **35** (5), 684–702 (2009)
4. Cubranic, D., Murphy, G.C.: Hipikat: recommending pertinent software development artifacts. In: Proceedings of the 25th International Conference on Software Engineering (ICSE '03), pp. 408–418 (2003)
5. Cubranic, D., Murphy, G.C., Singer, J., Booth, K.S.: Hipikat: a project memory for software development. IEEE Trans. Softw. Eng. **31**(6), 446–465 (2005)
6. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. J. Softw. Maint. Evol. **19**(2), 77–131 (2007)
7. Kilpinen, M.S.: The emergence of change at the systems engineering and software design interface: an investigation of impact analysis. Ph.D. thesis, University of Cambridge, Cambridge, UK (2008)
8. Lehnert, S.: A taxonomy for software change impact analysis. In: Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution (IWPSE-EVOL 2011), Szeged, Hungary, September 2011, pp. 41–50 (2011)
9. Lehnert, S.: A review of software change impact analysis. Technical report, Technische Universität Ilmenau (2011). URN: urn:nbn:de:gbv:ilm1-2011200618
10. Li, B., Sun, X., Leung, H., Zhang, S.: A survey of code-based change impact analysis techniques. Softw. Test. Verification Reliab. **23**, 613–646 (2013)
11. Lindvall, M., Sandahl, K.: How well do experienced software developers predict software change? J. Syst. Softw. (archive) **43**(1), 19–27 (1998). Elsevier Science Inc., New York, NY, USA
12. Van de Laar, P.: Transferring evolutionary couplings to industry. In: Van de Laar, P., Punter, T. (eds.) Views on Evolvability of Embedded Systems, pp. 69–88. Embedded Systems. Springer, Rotterdam (2011)
13. Pirklbauer, G., Fasching, Ch., Kurschl, W.: Improving change impact analysis with a tight integrated process and tool. In: 7th International Conference on Information Technology: New Generations (ITNG 2010), Las Vegas, Nevada, USA, April 2010, pp. 12–14 (2010)
14. Ratzinger, J., Fischer, M., Gall, H.: EvoLens: lens-view visualizations of evolution data. In: Proceedings of the 8th International Workshop on Principles of Software Evolution (IWPSE '05), pp. 103–112. IEEE Computer Society (2005)
15. Ren, X., Shah, F., Tip, F., Ryder, B.G., Chesley, O.: Chianti: a tool for change impact analysis of java programs. In: Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '04), pp. 432–448 (2004)
16. Rijsbergen, C.J.V.: Information Retrieval. Butterworths, London (1979)
17. Sliwerski, J., Zimmermann, T., Zeller, A.: HATARI: raising risk awareness. In: Proceedings of the 10th European Software Engineering Conference, 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE), Lisbon (2005)
18. Tóth, G., Hegedűs, P., Beszédes, Á., Gyimóthy, T., Jász, J.: Comparison of different impact analysis methods and programmer's opinion: an empirical study. In: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java (PPPJ '10), New York, NY, USA, pp. 109–118 (2010)

19. Wetzlmaier, T., Klammer, C., Ramler, R.: Extracting dependencies from software changes: an industry experience report. In: Proceedings of the 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement (IWSM-Mensura). IEEE Computer Society (2014)
20. Zimmermann, T., Zeller, A., Weißgerber, P., Diehl, S.: Mining version histories to guide software changes. IEEE Trans. Softw. Eng. **31**(6), 429–445 (2005). IEEE Press

Heterogeneous Systems Testing Techniques: An Exploratory Survey

Ahmad Nauman Ghazi(✉), Kai Petersen, and Jürgen Börstler

Blekinge Institute of Technology, Karlskrona, Sweden
{nauman.ghazi,kai.petersen,jurgen.borstler}@bth.se

Abstract. Heterogeneous systems comprising sets of inherent subsystems are challenging to integrate. In particular, testing for interoperability and conformance is a challenge. Furthermore, the complexities of such systems amplify traditional testing challenges. We explore (1) which techniques are frequently discussed in literature in context of heterogeneous system testing that practitioners use to test their heterogeneous systems; (2) the perception of the practitioners on the usefulness of the techniques with respect to a defined set of outcome variables. For that, we conducted an exploratory survey. A total of 27 complete survey answers have been received. Search-based testing has been used by 14 out of 27 respondents, indicating the practical relevance of the approach for testing heterogeneous systems, which itself is relatively new and has only recently been studied extensively. The most frequently used technique is exploratory manual testing, followed by combinatorial testing. With respect to the perceived performance of the testing techniques, the practitioners were undecided regarding many of the studied variables. Manual exploratory testing received very positive ratings across outcome variables.

1 Introduction

Over the years, software has evolved from simple applications to large and complex system of systems [8]. A system of systems consists of a set of individual systems that together form a new system. The system of systems could contain hardware as well as software systems. Recently, system of systems has emerged as a highly relevant topic of interest in the software engineering research community investigating its implications for the whole development life cycle. For instance, in the context of system of systems, Lane [20] studied the impact on development effort, Ali et al. [2] investigated testing, and Lewis et al. [22] proposed a process of how to conduct requirements engineering.

Systems of systems often exhibit heterogeneity [21], for instance, in implementation, hardware, process and verification. For the purpose of this study, we define a *heterogeneous system* as a system comprised of multiple systems (system of systems) where at least one subsystem exhibits heterogeneity with respect to the other systems [13]. Heterogeneity may occur in terms of different system complexities [33], programming languages and platforms [44], types of systems [10], system supply (internal vs. third party) [19], development processes and

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 67–85, 2015.
DOI: 10.1007/978-3-319-13251-8_5

distribution among development sites [15]. However, no frameworks are available as yet to measure these heterogeneity dimensions. The system of systems approach taken in development of heterogeneous systems give rise to various challenges due to continuous change in configurations and multiple interactions between the functionally independent subsystems. The challenges posed to testing of heterogeneous systems are mainly related to interoperability [28,44], conformance [28] and large regression test suites [2,6]. Furthermore, the inherent complexities of heterogeneous systems also pose challenges to the specification, selection and execution of tests.

In recent years, together with the emergence of system of systems research testing of heterogeneous systems received an increased attention from the research community. However, solutions proposed have been primarily evaluated from the academic perspective, and not the viewpoint of the practitioner.

In this study, we explored the viewpoint of practitioners with respect to testing heterogeneous systems. Two main contributions are made:

C1: *Explore which testing techniques investigated in research are used by practitioners.* Thereby, we learn which techniques practitioners are aware of, and which ones are most accepted.

C2: *Explore the perception of the practitioners of how well the used techniques perform with respect to a specified and frequently studied set of outcome variables.* Understanding the practitioners' perception of the techniques relative to each other allows to identify preferences from the practitioners' viewpoint. The findings will provide interesting pointers for future work to understand the reasons for the findings, and improve the techniques accordingly.

The contributions are made by using an exploratory survey to capture the opinion of practitioners.

The remainder of the paper is structured as follows: Sect. 2 presents the related work. Section 3 outlines the research method, followed by the results in Sect. 4. Section 5 presents a discussion of observations from the results. Finally, in Sect. 6, we conclude this study.

2 Related Work

The related work focuses on testing of heterogeneous systems, first discussing testing of heterogeneous systems as such, followed by reviewing solutions of how to test them. However, no surveys could be found that discuss any aspect of testing of heterogeneous systems.

2.1 Testing in Heterogeneous Systems

Testing heterogeneous systems is primarily considered to be a challenge emanating from the problem of integration and system-level testing [9,41]. Therefore, the current research in the area of heterogeneous systems considers it as

a subsystem interaction issue [41]. It is also observed that solving the inherent complexities underlying the testing heterogeneous systems is not a priority. Therefore, most of the related research is focused on addressing the accidental complexities in testing of heterogeneous systems by tuning and optimizing different testing techniques and methods.

A number of research studies discuss system-level testing in general terms without addressing specific test objectives. For automated functional testing, Donini et al. [9] propose a test framework where functional testing is conducted in an external simulated environment based on service-oriented architectures. The researchers demonstrated that functional system testing through simulated environments can be an approach to overcome the challenge of minimizing test sets. The test sets identified were representative of the real operational usage profile for the system. Wang et al. [41] study heterogeneous systems that exhibit heterogeneity at the platform level and discussed different factors considered in system-level testing of heterogeneous systems. Other than the studies focusing on system and integration testing, a relatively small set of studies attempt to discuss the problem of testing in heterogeneous systems in other test phases. Mao et al. [23] study this problem in the unit test phase whereas Diaz [7] addresses the problem of testing heterogeneous systems in the acceptance testing phase.

Research literature related to testing of heterogeneous systems frequently discusses the interoperability as a common issue. Interoperability testing is also a key test objective in different applications and technology domains. Xia et al. [44] address the interoperability problem in the web service domain and propose a test method to automate conformance and interoperability testing for e-business specification languages. Narita et al. [28] propose a method supported by a testing framework for interoperability testing for web service domain focusing on communication in robotics domain. However, interoperability remains a challenge in other domains as well. In context of large scale component based systems, Piel et al. [35] present a virtual component testing technique and demonstrated how virtual components can be formed using three different algorithms. This technique was further implemented and evaluated in industrial settings. Furthermore, Kindrick et al. [18] propose a technique combining interoperability testing with conformance testing and conclude that combining the two techniques will reduce the cost of setting up and executing the test management processes improving the effectiveness.

2.2 Testing Techniques

We surveyed techniques on testing heterogeneous systems that have been reported to be used in literature or industry for that purpose. These have been identified through a systematic literature review and a case study [12] and represent three coarse categories of testing techniques, namely manual exploratory, combinatorial, and search-based testing.

Manual Exploratory testing: Manual exploratory testing (ET) is an approach to test software without pre-defined test cases in contrast with traditional

test case based testing. The main characteristics of exploratory testing are simultaneous learning, test design and execution [16,40]. The tester has the freedom to dynamically design, modify and execute the tests.

In past, exploratory testing was seen as an ad-hoc approach to test software. However, over the years, ET has evolved into a more manageable and structured approach without compromising the freedom of testers to explore, learn and execute the tests in parallel. An empirical study comparing the effectiveness of exploratory testing with test-case based testing was conducted by Bhatti and Ghazi [4] and further extended (cf. [1]). This empirical work concludes that ET produces more defects as compared to test case based testing where time to test is a constraint.

Combinatorial Testing: Combinatorial testing is used to test applications for different test objectives at multiple levels. A comprehensive survey and discussion is provided by Nie and Leung [29]. It has been used for both unit and system-level testing in various domains. Combinatorial testing tends to reduce the effort and cost for effective test generation [5]. There exist a number of variants of combinatorial testing, which are used in different domains to test heterogeneous systems.

The problem of testing web services is the most common area in heterogeneous systems that is addressed in literature using different test techniques as discussed in Sect. 2.1. Mao et al. [23] and Apilli [3] proposed different frameworks for combinatorial testing to test component based software systems in a web services domain.

Wang et al. [42] study the problem of how interaction faults can be located based on combinatorial testing rather than manual detection and propose a technique for interactive adaptive fault location. Results from this study show that the proposed technique performs better than the existing adaptive fault location techniques.

Changing configurations pose challenges to combinatorial testing techniques. To that end Cohen et al. [6] conducted an empirical study to quantify the effectiveness of test suites. The study shows that there is an exponential growth of test cases when configurations change and subsets of test suites are used, similar to what is common in regression testing.

Mirarab et al. [27] conducted an industrial case study and propose a set of techniques for requirement-based testing. The SUT was software for a range of wireless, mobile devices. They propose a technique to model requirements, a technique for automated generation of tests using combination strategies, and a technique for prioritization of existing test cases for regression testing.

Search-Based Software Testing: In search-based testing meta-heuristics are used to solve software testing problems by using, for example genetic algorithms, to search for a solution for a problem (e.g. to generate test data).

Marin et al. [24] present an integrated approach where search-based techniques are applied on top of more classical techniques to derive optimal test configurations for web applications. The authors describe state of art and future web applications as complex and distributed, exhibiting several dimensions of heterogeneity. The study describes an approach that integrates combinatorial

testing, concurrency testing, oracle learning, coverage analysis, and regression testing with search-based testing to generate test cases.

Shiba et al. [38], proposed two artificial life algorithms to generate minimal test sets for t-way combinatorial testing based on a genetic algorithm (GA) and an ant colony algorithm (ACA). Experimental results show that when compared to existing algorithms including AETG (Automatic Efficient Test Generator) [5], simulated annealing-based algorithm (SA) and in-parameter order algorithm (IPO), this technique works effectively in terms of size of test set as well as time to execute.

Another study by Pan et al. [31] explores search-based techniques and defines a novel algorithm, i.e., OEPST (organizational evolutionary particle swarm technique), to generate test cases for combinatorial testing. This algorithm combines the characteristics of organizational evolutionary idea and particle swarm optimization algorithm. The experimental results of this study show that using this new algorithm can reduce the number of test cases significantly.

There are refinements of exploratory, combinatorial, and search-based testing. However, these have not been surveyed to keep the questionnaire at a manageable length in order to avoid dropouts. Manual exploratory testing is a manual testing technique where the tester simultaneously learns, designs, and executes tests. The thought process (e.g. whether a specific technique inspires the test design) is not prescribed. Both combinatorial and search-based testing are usually supported by tools and automated, while they have different approaches in solving the testing problem (see above).

3 Research Method

The survey method used in this study is an exploratory survey. Thörn [39] distinguishes statistical and exploratory surveys.

In exploratory surveys the goal is not to draw general conclusion about a population through statistical inference based on a representative sample. A representative sample (even for a local survey) has been considered challenging, the author [39] points out that: *"This [remark by the authors: a representative sample] would have been practically impossible, since it is not feasible to characterize all of the variables and properties of all the organizations in order to make a representative sample."* Similar observations and limitations of statistical inference have been discussed by Miller [26].

Given that the focus of this research is specific to heterogeneous systems, the population is limited. We were aware of specific companies and practitioners that work with such systems, but the characteristics of companies and their products were not available to us. Hence, an exploratory survey was conducted to answer our research questions. Though, aim was to gather data from companies with different characteristics; different domains, sizes, etc. represented; for the obtained answers, external validity is discussed in Sect. 3.5.

3.1 Study Purpose

The goal of the survey is formulated based on the template suggested in [43] to define the goals of empirical studies. The goal for this survey is to explore *the testing of heterogeneous systems* with respect to the *usage and perceived usefulness of testing techniques used for heterogeneous systems* from the point of view of *industry practitioners* in the context of *practitioners involved in heterogeneous system development reporting their experience on heterogeneous system testing.*

In relation to the research goal two main research questions (RQs) were asked:

RQ1: *Which testing techniques are used to evaluate heterogeneous systems?*
RQ2: *How do practitioners perceive the identified techniques with respect to a set of outcome variables?*

3.2 Survey Distribution and Sample

We used convenience sampling to obtain the answers. Of interest were practitioners that were involved in the testing of heterogeneous systems before, thus not every software tester would be a suitable candidate for answering the survey. The sample was obtained through personal contacts as well as postings in software engineering web communities (e.g. LinkedIn and Yahoo Groups). 100 personal contacts were asked to respond, and to distribute the survey later. Furthermore, we posted the survey on 32 communities.

Overall, we obtained 42 answers, of which 27 were complete and valid. One answer was invalid as each response was given as "others", without any further specification. The remaining respondents did not complete the survey. We provide further details on the respondents and their organizations in Sect. 4.1.

3.3 Instrument Design

The survey instrument is structured along the following themes[1].

- *Respondents:* In this theme information about the respondent is collected. This information is comprised of: current position; duration of working in the current position in years; duration of working with software development; duration of working with testing heterogeneous systems.
- *Company, processes, and systems:* This theme focuses on the respondents' organizations and the characteristics of the products.
- *Test coverage:* Here the practitioners rate the importance of different coverage criteria on a 5-point Likert scale from *"Very Important"* to *"Unimportant"*. The coverage criteria rated were specification-based, code-based, fault-based, and usage-based.
- *Usage of testing techniques:* We identified three categories of testing techniques through our ongoing systematic literature review that have been attributed and used in testing heterogeneous systems, namely search-based,

[1] The survey can be found at https://www.surveymonkey.com/s/RP6DQKF.

Table 1. Surveyed variables

Variable	References
Ease of use	[17,34]
Effectiveness in detecting critical defects	[1]
Number of false positives	[1]
Effectiveness in detecting various types of defects	[1]
Time and cost efficiency	[1,34]
Effectiveness in detecting interoperability issues	[32]
Effectiveness for very large regression test sets	[14]
External product quality	[30]

combinatorial, and manual exploratory testing (see also Sect. 2). The concepts of the testing techniques were defined in the survey to avoid any confusion. Two aspects have been captured, usage and evaluation. With respect to usage we asked for the frequency of using the different techniques on a 7-point Likert scale ranking from *"Always"* to *"Never"*. We also provided the option *"Do not know the technique"*.

- *Usefulness of testing techniques:* Each technique has been rated according to its usefulness with respect to a set of outcome variables that are frequently studied in literature on quality assurance techniques. The usefulness for each technique for each variable was rated on a 5-point Likert scale from *"Strongly Agree"* to *"Strongly Disagree"*. Table 1 provides an overview of the studied variables and their definitions.
- *Contact details:* We asked the respondents for their company name and e-mail address. The answer to this question was optional in case the respondents wished to stay anonymous towards the researchers.

The design of the survey has been pretested by three external practitioners and one researcher. The feedback led to minor reformulation and changes in the terminology used to become clear for practitioners. Furthermore, the number of response variables has been reduced to make the survey manageable in time and avoid maturation. Furthermore, the definition of heterogeneous system was revised to be more understandable. We further measured the time the respondents needed in the pretest to complete the survey. The time was between 10 and 15 min.

3.4 Analysis

For reflection on the data (not for inference) we utilized statistical tests to highlight differences for the techniques surveyed across the outcome variables. The Friedman test [11] (non-parametric test) has been chosen given multiple variables (treatments) were studied, the data being on ordinal scale.

3.5 Validity Threats

Internal Validity. One threat to capturing truthfully is if the questions asked in the survey are misunderstood. To reduce this threat we pretested the survey and made updates based on the feedback received. Another threat is maturation where the behavior changes over time. This threat has been reduced by designing the survey so that no more than 15 min were necessary to answer the survey.

Construct Validity. Theoretical validity is concerned with not being able to capture what we intend to capture (in this case the usefulness of different techniques across different outcome variables). To reduce this threat we defined variables based on literature, in particular focusing on variables that are frequently studied when evaluating quality assurance approaches. Given that the study is based on the subjects' experience, the lack of experience in search-based testing limits the comparability, given that eight respondents did not know the technique, and five have never used it. However, the remaining respondents had experience using it. For the other techniques (manual exploratory testing and combinatorial testing) only few respondents did not know them, or lacked experience. Given that the aim of the study is not to generalize the findings through inference, but rather identify interesting patterns and observations in an exploratory way, threats related to statistical inference were not emphasized.

External Validity. The exploratory nature of the survey does not allow to statistically generalize to a population. However, as suggested by [39], interesting qualitative arguments can be made such studies. The context captured in the demographics of the survey limits the external generalizability. In particular, the majority of respondents were related to the consulting industry (35.7 %), followed by computer industry (28.6 %), and communications (25.0 %), other industries only have very few responses and are not represented in this study (e.g. accounting, advertising, etc.). With regard to company size, all four size categories are equally well represented. With regard to development models, agile and hybrid processes have the highest representation. The data is hence not relevant for the other models. Overall, the external validity could be strengthened by a higher number of answers. Though, given that the survey was focused on heterogeneous systems the possible sample was reduced. In comparison, with a similar strategy of distributing a survey on a wider topic (automated software testing) over 100 valid responses could be obtained [36].

Conclusion Validity. Interpretive validity is primarily concerned with conclusions based on statistical analysis, and researcher bias when drawing conclusions. Given that the involved researchers have no particular preference on any of the solutions surveyed based on previous research, this threat can be considered as being under control.

4 Results

We first describe the study context as this allows companies to compare their own context, and hence being able to determine to what degree the results are

Table 2. Roles of subjects

Responsibility	Percent	Responses
Software developer (implementation, coding etc.)	22.2	6
Software architect (software structure, architecture, and design)	18.5	5
Software verification & validation (testing, inspection, reviews etc.)	18.5	5
Software quality assurance (quality control, quality management etc.)	14.8	4
System analyst (requirements elicitation, analysis, specification and validation etc.)	7.4	2
Project manager (project planning, project measurement etc.)	3.7	1
Product manager (planning, forecasting, and marketing software products etc.)	0.0	0
Software process engineer (process implementation and change, process and product measurement etc.)	0.0	0
Other	11.1	3

relevant for them. Thereafter, we characterize the heterogeneity dimensions of the systems being reported by the practitioners. Thereafter, the answers to the research questions are presented.

4.1 Context

Subjects. Table 2 provides an overview of the primary roles of the subjects participating in the survey. The roles most frequently presented are directly related with either quality assurance, or the construction and design of the system. Overall, the experience in years in the current role indicates a fair to strong experience level of the respondents in their current positions.

Looking at the overall experience related to software engineering in years, the average experience is 10.55 years with a standard deviation of 7.04. This indicates that the overall experience in software development is very high.

We also asked for the experience of the practitioners in testing heterogeneous systems themselves. The average experience in testing heterogeneous systems is 4.63 years with a standard deviation of 5.22, while 8 respondents did not have experience as testers on heterogeneous systems themselves. The survey focused on practitioners involved in developing heterogeneous systems though, as those also often gain insights on the quality assurance processes (e.g. people in quality management). Hence, those responses were not excluded.

Company, processes, and systems. The number of responses in relation to company size are shown in Table 3. All sizes are represented well by the respondents, hence the results are not biased towards a specific company size.

The companies surveyed worked in 24 different industry sectors (one company can work in several sectors, hence multiple answers were possible).

Table 3. Company size (number of employees)

Size (no. of employees)	Percent	Responses
Less than 50	18.5	5
50 to 249	29.6	8
250 to 4499	29.6	8
5400 and more	22.2	6

Table 4. System types

System type	Percent	Responses
Data-dominant software	63.0	17
Control-dominant software	25.9	7
Computation-dominant software	25.9	7
Systems software	22.2	6
Other	14.8	4

Table 5. Development models

Model	Percent	Responses
Agile	29.6	8
Hybrid process (dominated by agile practices, with few plan-driven practices)	29.6	8
Waterfall	11.1	3
V-Model	11.1	3
Hybrid process (dominated by plan-driven practices, with few agile practices)	11.1	3
Spiral	3.7	1
Other	7.4	2

The industries that were represented by the highest number of respondents were consulting (9 respondents), computer industry (hardware and desktop software) (7 respondents), communications (6 respondents), and business/professional services (5 respondents).

The systems developed are characterized by different types as specified in [10]. As shown in Table 4 the clear majority of respondents was involved in data-dominant software development, though all types were represented through the surveyed practitioners.

The development models used in the surveyed companies are illustrated in Table 5. The clear majority of respondents is working with agile development and hybrid processes that are dominated by agile practices.

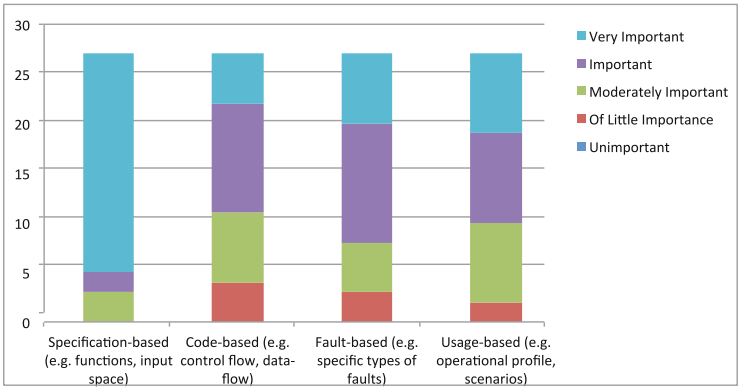

Fig. 1. Importance of test objectives

Test coverage. A key aspect of testing is the test objectives that drive the selection of test cases (cf. [25]). We captured the objectives of the participating industry practitioners in their test case selection as shown in Fig. 1. Specification-based coverage is clearly the most important criterion for the studied companies, followed by fault-based coverage. Overall, all coverage objectives are considered important by at least half of the participants.

4.2 Heterogeneity of Systems

There exist no agreement in literature on the definition of heterogeneity in systems. However, individual papers defined different dimensions of heterogeneity. In the survey, we asked the participants which dimensions are applicable to them, as illustrated in Table 6. The table shows that systems of different complexity, platforms and programming languages, and types of systems were the most common dimensions.

The respondents could select multiple heterogeneity items, as several may apply to their development context. For at least half of the systems the respondents selected three or more heterogeneity dimensions that apply to them.

Table 6. Heterogeneity dimensions in the studied systems

Heterogeneity dimensions	Percent	Responses
System complexity	70.4	19
Programming language and platforms	59.2	16
Type of systems	55.56	15
System supply (internally developed and third party)	48.1	13
Development processes	40.7	11
Distribution of development systems in different locations	25.9	7

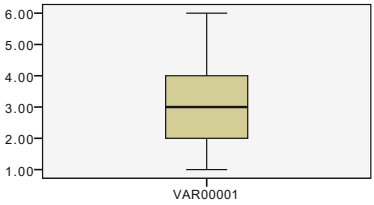

Fig. 2. Number of heterogeneity dimensions selected

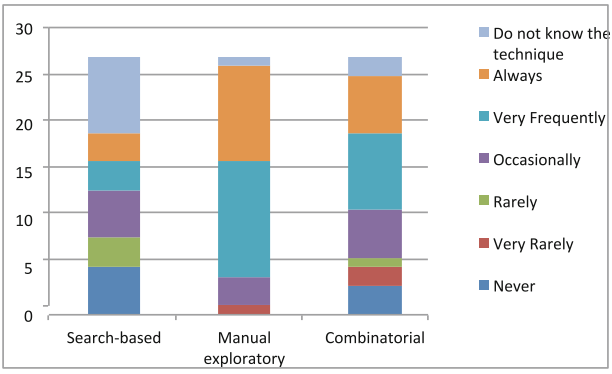

Fig. 3. Usage of techniques in heterogeneous systems

A quarter of all systems surveyed is characterized by four or more dimensions (see Fig. 2). Only few systems are only characterized by one of the dimensions.

4.3 RQ1: Usage of Testing Techniques

We captured the frequency of usage for the three different techniques introduced earlier (search-based, manual exploratory, and combinatorial testing). The frequencies are illustrated in Fig. 3.

Looking at the overall distribution of usage, it is clearly visible that manual exploratory testing is the most frequently used technique, followed by combinatorial testing and search-based testing. There was not a single respondent indicating of never having used manual exploratory testing.

Search-based testing is the least-used technique, as well as the technique that is least-known. However, 3 respondents who mentioned that they always use search-based testing are all test consultants. Another consultant mentioned frequent usage of the technique along with 2 more respondents who are in education and professional services industries, respectively. Only very few respondents are not aware of manual exploratory and combinatorial testing, while the usage appears to depend on the role of the respondent.

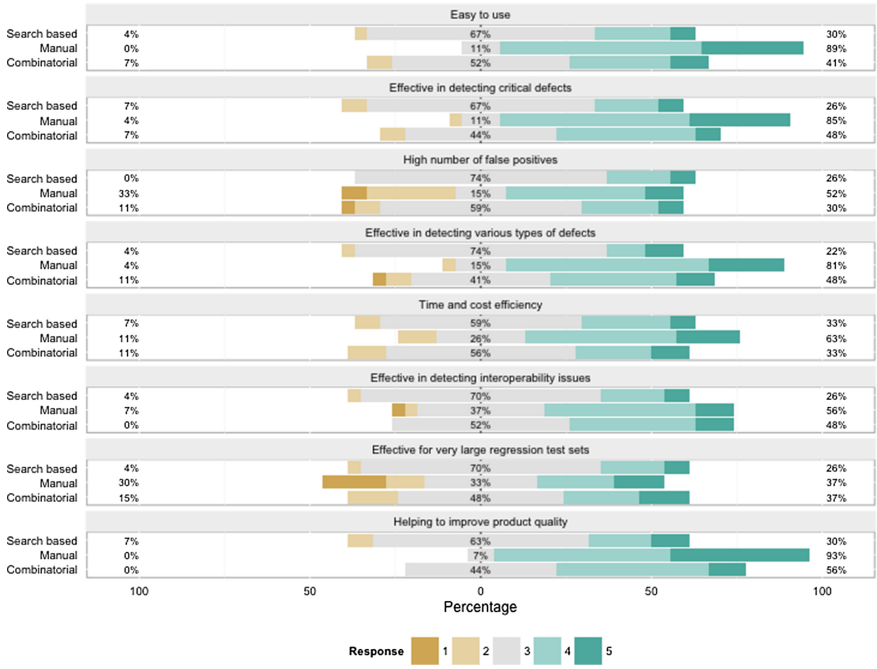

Fig. 4. Practitioners' perceptions of testing techniques for heterogeneous systems (1 = Strongly Disagree, 2 = Disagree, 3 = Uncertain, 4 = Agree, 5 = Strongly Agree)

4.4 RQ2: Perceived Usefulness

Figure 4 provides the rating of the variables for the three different techniques studied. To highlight patterns in the data, we also used statistical testing as discussed in Sect. 3.4. The results of the test are shown in Table 7.

The highest undecided rates are observed for search-based testing. This can be explained by the observation that people were not aware of the technique, or never used it (see Fig. 3). Also, a relatively high undecided rate can be seen for combinatorial testing, however, this cannot be attributed to the lack of knowledge about the technique, or that practitioners never used it, as the numbers on both items were relatively low. The opposite is true for manual exploratory testing, where only very few practitioners were undecided.

Variables that are more unique and emphasized for heterogeneous systems (effectiveness in detecting interoperability issues and effectiveness for very large regression test sets) have higher undecided rates for all the techniques. That is, there is a high level of uncertainty across techniques. In the case of regression tests manual exploratory testing was perceived as the most ineffective. For interoperability testing no major difference between the ratings can be observed, which is also indicated by the statistical tests shown in Table 7.

Of all techniques, manual exploratory testing is rated exceptionally high in comparison to other techniques for ease of use, effectiveness in detecting critical

Table 7. Friedman test statistics

Item	N	χ^2	df	p-value
Easy to use	27	22.522	2	0.000
Effective in detecting critical defects	27	19.500	2	0.000
High number of false positives	27	0.090	2	0.956
Effective in detecting various types of defects	27	17.848	2	0.000
Time and cost efficiency	27	3.797	2	0.150
Effective in detecting interoperability issues	27	7.000	2	0.030
Effective for very large regression test sets	27	1.509	2	0.470
Helping to improve product quality	27	25.400	2	0.000

defects, detecting various types of defects, and in improving product quality. The high rating is also highlighted through the statistical tests, which detected this as a difference in the data sets (see Table 7). At the same time, it also received the strongest negative ratings, which was the case for false positives and effectiveness for very large regression test suites.

5 Discussion

Based on the data collected, we highlight interesting observations, and present their implications.

Observation 1: Interestingly, search-based testing was applied by several practitioners in the scoped application of testing heterogeneous systems (in total 14 of 27 used it at least very rarely), even though in comparison it was the least frequently applied technique. Literature surveying research on search-based testing reported acknowledges that testing is primarily a manual process [25]. Also, in heterogeneous systems we only identified few studies in our search for literature that used search-based testing. Hence, it is an interesting observation that companies are using search-based testing. At the same time, many practitioners were not aware of it at all. This leads to the following lessons learned:

> **Lessons learned:** First, given the presence of search-based testing in industry, there exist opportunities for researchers to study it in real industrial environments and to collect experiences made by practitioners; Second, practical relevance of search-based testing in heterogeneous testing is indicated by the adoption of the technique, which is encouraging for this relatively new field.

Observation 2: Although, the survey was targeted towards a specific group of practitioners that have experience with developing and testing heterogeneous

systems, the practitioners were largely undecided on whether the techniques used are suitable for detecting interoperability issues. Figure 4 shows that search-based testing has comparatively high undecided rates for all the variables.

> **Lessons learned:** Practitioners require further decision support and comparisons to be able to make informed decisions about the techniques given the high level of uncertainty. In particular, further comparative studies (which were lacking) are needed in general, and for heterogeneous systems in particular. If people are undecided, adoption is also hindered; hence one should aim to reduce the uncertainty on outcomes for the variables studied.

Observation 3: Manual exploratory testing is perceived as very positive by practitioners for the variables "Ease of use", "Effective in detecting critical defects", "Effective in detecting various types of defects", "Time and cost effective" and "Helping to improve product quality". On the other hand, it has been perceived poorly in comparison to other techniques for the variables "High number of false positives" and "Effective for very large regression-test suites". Given the context of testing heterogeneous systems, these observations are interesting to compare with findings of studies investigating exploratory testing. Shah et al. [37] investigated exploratory testing and contrasted the benefits and advantages of exploratory and scripted testing through the application of a systematic review combined with expert interviews. Their review is hence used as a basis for the comparison with literature.

The finding with respect to ease of use was understandable, but could also be seen as a paradox. On the one hand there are no perceived barriers as one does not have to learn testing techniques; however, the quality of tests is not known because there is such a high dependency on the skills of the testers (cf. Shah et al. [37]), which could potentially lead to a wrong perception. Shah et al. identified multiple studies indicating time and cost efficiency, and also confirmed that the exploratory testing is good at identifying the most critical defects. Overall, this appears to be well in-line with the findings for heterogeneous systems. With respect to false positives, the practitioners were in disagreement on whether manual exploratory testing leads to a high number of false positives. Literature on the other hand suggests that fewer false positives are found. With respect to regression testing, the findings indicate the potential for better regression testing in case that sessions are properly recorded, but it was also recognized that it is difficult to prioritize and reevaluate the tests.

> **Lessons learned:** Even though not representative, the data indicates a gap between industry focus and research focus. Therefore, research should focus on investigating exploratory testing, how it should be applied, and how efficient it is in capturing interoperability issues to support companies in improving their exploratory testing practices.

6 Conclusion

In this study we explored the testing of heterogeneous systems. In particular, we studied the usage and perceived usefulness of testing techniques for heterogeneous systems. The techniques were identified based on an ongoing systematic literature review. The practitioners surveyed were involved in the development of heterogeneous systems. Two main research questions were answered:

RQ1: Which testing techniques are used to assess heterogeneous systems? The most frequently used technique is exploratory manual testing, followed by combinatorial and search-based testing. As discussed earlier, it is encouraging for the field of search-based testing that a high number of practitioners have made experiences with search-based testing. This may provide opportunities to study the technique from the practitioners' perspective more in the future. Looking at the awareness, the practitioners were well aware of manual exploratory and combinatorial testing, however, a relatively high number was not aware of what search-based testing is.

RQ2: How do practitioners perceive the identified techniques with respect to a set of outcome variables? The most positively perceived technique for testing heterogeneous systems was manual exploratory testing, which was the highest rated in five (ease of use, effectiveness in detecting critical defects, effective in detecting various types of defects, time and cost efficiency, helping to improve product quality) out of eight studied variables. While manual exploratory testing was the most used technique in the studied companies, it is the least investigated technique in the literature on testing heterogeneous systems.

In future work, based on the results of the study, several important directions of research were made explicit:

- Given there are no frameworks available that can help identify, to what degrees one system is heterogeneous in comparison to other systems. Therefore, a framework will be provided to measure different dimensions of heterogeneity.
- In order to reduce the uncertainty with respect to the performance of the techniques comparative studies are needed. In particular, in the context of heterogeneous systems variables more relevant to that context should be studied (interoperability, large regression test suits). However, in general more comparative studies may be needed, for instance by comparing their performance on heterogeneous open source systems (e.g. Linux).
- Given the positive indications of the adoption of search-based in the industry, the focus should also be on understanding how and with what success search-based is used in the industry for heterogeneous and other systems.
- Interesting patterns identified and highlighted in the discussion should be investigated in further depth, two examples should be highlighted: First, does (and if so how) heterogeneity affect the performance of exploratory testing in terms of false positives reported? Second, how could it be explained that manual exploratory testing is so positively perceived? Possible propositions are there is a low perceived entry level of using the technique, while it is at the same time very hard to master given its dependence on the testers' skills.

Furthermore, interestingly it was perceived as being time- and cost efficient, which should be understood further. Overall, large and complex systems have many interactions that could require automation to be able to achieve a satisfactory level of coverage.

References

1. Afzal, W., Ghazi, A.N., Itkonen, J., Torkar, R., Andrews, A., Bhatti, K.: An experiment on the effectiveness and efficiency of exploratory testing. Empir. Softw. Eng. 1–35 (2014)
2. Ali, N.B., Petersen, K., Mäntylä, M.: Testing highly complex system of systems: an industrial case study. In: Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2012), pp. 211–220. ACM (2012)
3. Apilli, B.S.: Fault-based combinatorial testing of web services. In: Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, 25–29 October 2009, Orlando, Florida, USA, pp. 731–732 (2009)
4. Bhatti, K., Ghazi, A.N.: Effectiveness of exploratory testing: an empirical scrutiny of the challenges and factors affecting the defect detection efficiency. Master's thesis, Blekinge Institute of Technology (2010)
5. Cohen, D., Dalal, S., Fredman, M., Patton, G.: The AETG system: an approach to testing based on combinatorial design. IEEE Trans. Softw. Eng. **23**(7), 437–444 (1997)
6. Cohen, M.B., Snyder, J., Rothermel, G.: Testing across configurations: implications for combinatorial testing. Softw. Eng. Notes **31**(6), 1–9 (2006)
7. Diaz, J., Yague, A., Alarcon, P.P., Garbajosa, J.: A generic gateway for testing heterogeneous components in acceptance testing tools. In: Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), pp. 110–119, Feb 2008 (2008)
8. DoD. Systems and software engineering. systems engineering guide for systems of systems, version 1.0. Technical Report ODUSD(A&T)SSE, Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Washington, DC, USA (2008)
9. Donini, R., Marrone, S., Mazzocca, N., Orazzo, A., Papa, D., Venticinque, S.: Testing complex safety-critical systems in SOA context. In: 2008 International Conference on Complex, Intelligent and Software Intensive Systems, pp. 87–93 (2008)
10. Forward, A., Lethbridge, T.C.: A taxonomy of software types to facilitate search and evidence-based software engineering. In: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, p. 14. ACM (2008)
11. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc. **32**(200), 675–701 (1937)
12. Ghazi, A.N.: Testing of heterogeneous systems. Blekinge Institute of Technology Licentiate Dissertation Series 2014(03):1–153 (2014)
13. Ghazi, A.N., Andersson, J., Torkar, R., Petersen, K., Börstler, J.: Information sources and their importance to prioritize test cases in the heterogeneous systems context. In: Barafort, B., Messnarz, R., O'Connor, R.V., Poth, A. (eds.) EuroSPI 2014. CCIS, vol. 425, pp. 86–98. Springer, Heidelberg (2014)

14. Graves, T.L., Harrold, M.J., Kim, J.-M., Porter, A., Rothermel, G.: An empirical study of regression test selection techniques. In: Proceedings of the 20th International Conference on Software Engineering, ICSE '98, Washington, DC, USA, pp. 188–197. IEEE Computer Society (1998)
15. Herbsleb, J.D.: Global software engineering: the future of socio-technical coordination. In: 2007 Future of Software Engineering, pp. 188–198. IEEE Computer Society (2007)
16. Kaner, C., Bach, J., Pettichord, B.: Lessons Learned in Software Testing. Wiley, New York (2008)
17. Karahanna, E., Straub, D.W.: The psychological origins of perceived usefulness and ease-of-use. Inf. Manag. **35**(4), 237–250 (1999)
18. Kindrick, J.D., Sauter, J.A., Matthews, R.S.: Interoperability testing. Stand. View **4**(1), 61–68 (1996)
19. Kontio, J.: A case study in applying a systematic method for cots selection. In: Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 201–209. IEEE (1996)
20. Lane, J.A.: SoS management strategy impacts on SoS engineering effort. In: Münch, J., Yang, Y., Schäfer, W. (eds.) ICSP 2010. LNCS, vol. 6195, pp. 74–87. Springer, Heidelberg (2010)
21. Lewis, G., Morris, E., Place, P., Simanta, S., Smith, D., Wrage, L.: Engineering systems of systems. In: 2008 2nd Annual IEEE Systems Conference, pp. 1–6. IEEE (2008)
22. Lewis, G.A., Morris, E., Place, P., Simanta, S., Smith, D.B.: Requirements engineering for systems of systems. In: 2009 3rd Annual IEEE Systems Conference, pp. 247–252. IEEE (2009)
23. Mao, C.: Towards a hierarchical testing and evaluation strategy for web services system. In: 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications, pp. 245–252 (2009)
24. Marin, B., Vos, T., Giachetti, G., Baars, A., Tonella, P.: Towards testing future Web applications. In: 2011 Fifth International Conference on Research Challenges in Information Science (RCIS), May 2011, pp. 1–12 (2011)
25. McMinn, P.: Search-based software test data generation: a survey. Softw. Test. Verif. Reliab. **14**(2), 105–156 (2004)
26. Miller, J.: Statistical significance testing-a panacea for software technology experiments? J. Syst. Softw. **73**, 183–192 (2004)
27. Mirarab, S., Ganjali, A., Tahvildari, L., Li, S., Liu, W., Morrissey, M.: A requirement-based software testing framework : an industrial practice. Test, pp. 452–455 (2008)
28. Narita, M., Shimamura, M., Iwasa, K., Yamaguchi, T.: Interoperability verification for Web Service based robot communication platforms. In: IEEE International Conference on Robotics and Biomimetics, 2007, ROBIO 2007, pp. 1029–1034, December 2007
29. Nie, C., Leung, H.: A survey of combinatorial testing. ACM Comput. Surv. **43**(2), 1–29 (2011)
30. Ortega, M., Pérez, M., Rojas, T.: Construction of a systemic quality model for evaluating a software product. Softw. Qual. J. **11**(3), 219–242 (2003)
31. Pan, X., Chen, H.: Using organizational evolutionary particle swarm techniques to generate test cases for combinatorial testing. In: 2011 Seventh International Conference on Computational Intelligence and Security, December 2011, pp. 1580–1583 (2011)

32. Perumal, T., Ramli, A.R., Leong, C.Y., Mansor, S., Samsudin, K.: Interoperability among heterogeneous systems in smart home environment. In: IEEE International Conference on Signal Image Technology and Internet Based Systems, 2008, SITIS '08, pp. 177–186 (2008)
33. Petersen, K., Khurum, M., Angelis, L.: Reasons for bottlenecks in very large-scale system of systems development. Inf. Softw. Technol. **56**(10), 1403–1420 (2014)
34. Petersen, K., Rönkkö, K., Wohlin, C.: The impact of time controlled reading on software inspection effectiveness and efficiency: a controlled experiment. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '08, pp. 139–148. ACM, New York (2008)
35. Piel, É., Gonzalez-Sanchez, A., Gross, H.-G.: Built-in data-flow integration testing in large-scale component-based systems. In: Petrenko, A., Simão, A., Maldonado, J.C. (eds.) ICTSS 2010. LNCS, vol. 6435, pp. 79–94. Springer, Heidelberg (2010)
36. Rafi, D.M., Moses, K.R.K., Petersen, K., Mäntylä, M.: Benefits and limitations of automated software testing: systematic literature review and practitioner survey. In: 2012 7th International Workshop on Automation of Software Test (AST), pp. 36–42. IEEE (2012)
37. Shah, S.M.A., Gencel, C., Alvi, U.S., Petersen, K.: Towards a hybrid testing process unifying exploratory testing and scripted testing. J. Softw. Evol. Process **25**(3), 261–283 (2013)
38. Shiba, T.: Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In: Computer Software and Applications Conference (COMPSAC 2004) (2004)
39. Thörn, C.: Current state and potential of variability management practices in software-intensive SMEs: results from a regional industrial survey. Inf. Softw. Technol. **52**(4), 411–421 (2010)
40. Van Veenendaal, E., et al.: The Testing Practitioner. UTN Publishers, Den Bosch (2002)
41. Wang, D., Barnwell, B., Witt, M.B.: A cross platform test management system for the SUDAAN statistical software package. In: 2009 Seventh ACIS International Conference on Software Engineering Research, Management and Applications, pp. 237–244 (2009)
42. Wang, Z., Xu, B., Chen, L., Xu, L.: Adaptive interaction fault location based on combinatorial testing. In: 2010 10th International Conference on Quality Software, July 2010, pp. 495–502 (2010)
43. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B.: Experimentation in Software Engineering. Springer, Heidelberg (2012)
44. Xia, Q.-M., Peng, T., Li, B., wen Feng, Z.: Study on automatic interoperability testing for e-business. In: CiSE 2009, International Conference on Computational Intelligence and Software Engineering, Dec 2009, pp. 1–4 (2009)

Software and Systems Architectures

Integrating Heterogeneous Engineering Tools and Data Models: A Roadmap for Developing Engineering System Architecture Variants

Richard Mordinyi(✉), Dietmar Winkler, Florian Waltersdorfer, Stefan Scheiber, and Stefan Biffl

Christian Doppler Laboratory, Software Engineering Integration for Flexible Automation Systems, Vienna University of Technology, Vienna, Austria
{richard.mordinyi,dietmar.winkler,florian.waltersdorfer, stefan.scheiber,stefan.biffl}@tuwien.ac.at
http://cdl.ifs.tuwien.ac.at

Abstract. Developing large systems engineering projects require combined efforts of various engineering disciplines. Each engineering group uses specific engineering tools and data model concepts representing interfaces to other disciplines. However, individual concepts lack in completeness and include strong limitations regarding interoperability and data exchange capabilities. Thus, highly heterogeneous data models cause semantic gaps that hinder efficient collaboration between various disciplines. The design of an integration solution within a systematic engineering process typically requires re-modelling of the common data model (used for mapping individual local tool data models) to enable efficient data integration. However, designing and implementing integration approaches include continuously collecting new knowledge on the related application domains, in our case automation systems engineering projects, and integration capability that meet requirements of related domains. In this paper we report on a sequence of different architectural designs for an efficient and effective integration solution that lead to a similar and stable data model design for application in the automation systems domain. By means of iterative prototyping, candidates for modelling styles were tested for feasibility in context of industry use cases. In addition we applied an adjusted Architecture Tradeoff Analysis Method (ATAM) to assess the resulting final architecture variant.

Keywords: Semantic integration · Data modelling · Service design · Service modelling

1 Introduction

The development of today's automation systems, like power plants or steel mills, requires the combined efforts of engineers from several disciplines such as software, electrical, and mechanical engineering. Software tools used by engineers

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 89–107, 2015.
DOI: 10.1007/978-3-319-13251-8_6

are rarely interoperable [9] and focus on distinct tasks for each engineering discipline. Related data models are often highly heterogeneous, causing so-called "semantic gaps" between disciplines and groups of engineers. In addition, models used in individual engineering disciplines and their best-practice tools often apply a range of different terms and/or modelling structures to describe a given data model concept leading to semantically heterogeneous models on project and team level [36]. In order to handle integration concerns at the interfaces of different engineering disciplines [3,4], common concepts aims at bridging the gap between different disciplines on project and team level. Exchanging data between disciplines and different engineering groups needs to be done "manually", as parts of the common data model are simply missing in some tools. These missing parts have to be added either via custom fields in the specific tool (i.e., fields with no special meaning to that tool) or have to be added after data exchange by editing export/import files "outside" of the tools. These tasks are typically executed by experts who are familiar with at least two related tools.

In an ongoing project[1] we focus on the development of different integration strategies in the automation systems domain that help engineers and managers in integrating data from heterogeneous engineering environments coming from various sources. In this paper we report on a series of architecture approaches and discuss lessons learned of individual architecture variants with respect to feasiblity, flexiblity, and applicabilty in heterogeneous engineering environments. Derived design guidelines - aligned with architecture variants - can support decision makers by enabling to (a) determine whether individual architecture variants are suitable for specific scenarios, (b) ease the initial modelling effort by avoiding known mistakes, and (c) enable later modifications of the data models without breaking the initial design. Finally, we evaluated the resulting "final" architecture variant with the adapted Architecture Tradeoff Analysis Method (ATAM) [21], a well-established approach for architecture evaluation based on requirements and scenarios. The resulting architecture was found useful in context of integration challenges in automation systems engineering projects with a heterogeneous set of tools and data models.

The remainder of this work is structured as follows: Sect. 2 presents related work on integration issues and data model approaches, Sect. 3 introduces to the industry use case in the automation systems domain and Sect. 4 presents the research issues. We report on the data model candidates and lessons learned of individual approaches in Sect. 5. Section 6 includes the evaluation results of the final architecture variant based on ATAM. Finally, Sect. 7 discusses the results, concludes the paper, and identifies future work.

2 Related Work

This section summarizes related work on integration scenarios and data integration approaches and introduces to the concept of "Enterprise Application Integration"

[1] Christian Doppler Laboratory "Software Engineering Integration for Flexible Automation Systems", http://cdl.ifs.tuwien.ac.at.

(EAI) and related state-of-the-art approaches required to design and evaluate software architectures.

2.1 Enterprise Application Integration

Enterprise Application Integration (EAI) approaches focus on four levels [23], i.e., Data Level, Application Interface Level, Method Level, and User Interface Level.

Integration on a **Data Level** refers to a scenario in which the "integrated" applications are not addressed directly by the integration solution, but via their data sources. Using the approach usually requires a very detailed understanding of a tool's database schemas, how the database is used during runtime and when/how it is safe to access. As stated in [13], data level integration is considered as the most basic form of application integration and can serve as a starting point for future integration efforts [14,42].

Application Interface Level Integration refers to using external access (API) features, provided by tools and tool vendors, to link applications and extend their capabilities as current software development best practices highly embrace open and reusable software architectures [11]. In contrast to Data Level Integration, this approach allows a more stable tool integration [7].

Method Level Integration refers to invasive coupling of applications in the way that internal methods and/or functionalities of software component are exposed to outside tools to enable an integrated solution. Because this approach requires a close cooperation of partners and most likely frequent refactoring steps to meet (evolving) integration requirements, this strategy may not be efficient in all cases [23]. However, this approach is strongly motivated around the idea of software reuse [25,27], and composition [2]. Following the DRY-principle [39] on a higher level, Method Level Integration aims at exploiting tool features to couple several different tools more efficient. Some research effort has been conducted to support [20,40] and measure [12] the development of extendible software with "clean" architectures [38] to make integration more feasible compared to earlier stages of software and data integration approaches [15].

Finally, the goal of **User Interface Level Integration** is not to connect parts of the software into larger systems, but to simulate user interaction or forward user interaction to the software component [30]. Thus, the "connected" software is treated as an isolated system to a much higher degree compared to other integration approaches.

2.2 Data Modeling and Integration

Designing and implementing integration solutions, especially in an enterprise environment, is a complex process with little to no standard solutions available. Yet there exists a series of guidelines, common terminology and well-defined patterns to deal with most known scenarios on an abstract level [16,17] as well as with concrete technological environments [18,26]. However, the degree of abstraction of data integration leaves much room for details about how to

connect and/or combine heterogeneous data sources. Outside the scope of EAI, works dealing with data integration separately, also retain a relatively high level of abstraction, either focusing on the granularity and frequency of the data integration and its use [33,37] and on the technical means to access and combine data sources [1]. In [35][2] a brief introduction to data integration styles, illustrating the connection between application and data integration, is given.

2.3 Data Modelling Styles

There are three commonly known and accepted styles about the basic structure of an integration model, with the first two defined in the early stages of data integration [8,22] and a third one which was added as an extension/advancement later on [24,28].

The basic assumption behind **Local as View** is that all data sources with their local schemas are more prone to change then the global schema against which they have to be integrated. Source models are treated as variants of a global model, allowing for an easy extension of the system, as long as the global schema remains "intact". The downsides of this style comes to light if either the global schema is unstable (i.e. due to changing integration requirements) or if the local sources are the true drivers for a data model.

The **Global as View** assumes that all data sources with their local schemas are more stable then the global schema against which they have to be integrated. All parts of the global model are treated as variants of a local model, allowing for an easy aggregation of the data from many local sources, as they are already configured in the mappings. Adding a new type of local source model to a GAV-modelled integration solution however, can result in a much higher effort.

In **Global-Local as View** both the local and the global model are treated as views (regardless if one or both are stored as actual data base instances) in the way that each element which is query-able from one of the views can be queried from the other view as well. This is achieved by using the model for GLAV itself to express the mappings between the local sources and the global mediation scheme. The main downside of this approach lies in its much higher complexity to model, implement and maintain a solution.

2.4 Architecture Evaluation with ATAM

The Architecture Tradeoff Analysis Method (ATAM) [21] provides for a framework to analyze, validate and improve software architecture designs ("styles") against a given set of requirements and how to derive said requirements (i.e. "quality attribute characterization"). With offering a more holistic approach for architecture analysis, ATAM has been evaluated extensively [34] and been applied to a variety of fields [5,10,19,31,32].

The ATAM core process consists of nine steps 1 separated into four stages:

[2] Downloadable http://cdlflex.org/conf/swqd15.

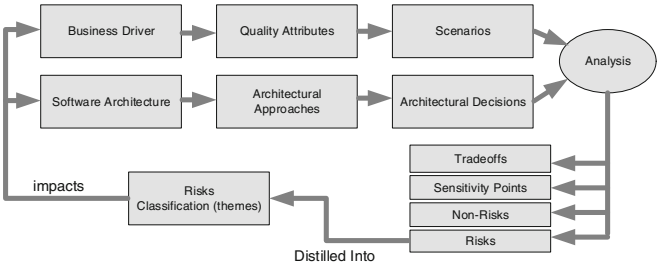

Fig. 1. The nine steps of the ATAM [21]

1. Presentation - consisting of steps 1–3 to present the ATAM itself, the business goals of the project and the current architecture
2. Investigation and Analysis - with steps 4–6 to analyze the architecture of the system under review, formulate current assumptions based on the business goals and evaluate the system under that knowledge
3. Testing - during steps 7 & 8 a larger set of stakeholders is invited to correct the business goal assumptions by comparing them to use cases and re-evaluating the system
4. Reporting - in step 9 to summarize and distribute the results of the ATAM execution

3 Use Case

The common use case in a multi-disciplinary engineering project is the review process, called "Signal Deletion with Review" involving electrical and software engineers.

While the Electrical Engineer (EE) works on the Circuit Design and uses a tool like EPlan[3], software engineers (SE) use logi.CAD[4] for control software programming. Those two programs use different data models to store their data, but they are mapped via a common data model, called "signals" (referring to the I/O-pins of the hardware or the respective variables in the software). The structure of these signals is a simple list of key-value pairs. The transformation from the original representation of the values to the common data model is either provided by the program itself, a script or by a connector that links the program to the information system. As the engineers collaborate in a project, this means that their individual work can affect the work of others, even if they are not changing someone else's data directly.

In case the electrical engineer (EE) modifies some local wire circuit design, the change made within the tool demands a propagation of modified data to the integration solution, which triggers the review process (see Fig. 2). If the modification done by EE does not cause any signal deletions (that is, all changes

[3] http://www.eplan.us
[4] http://www.logicals.com

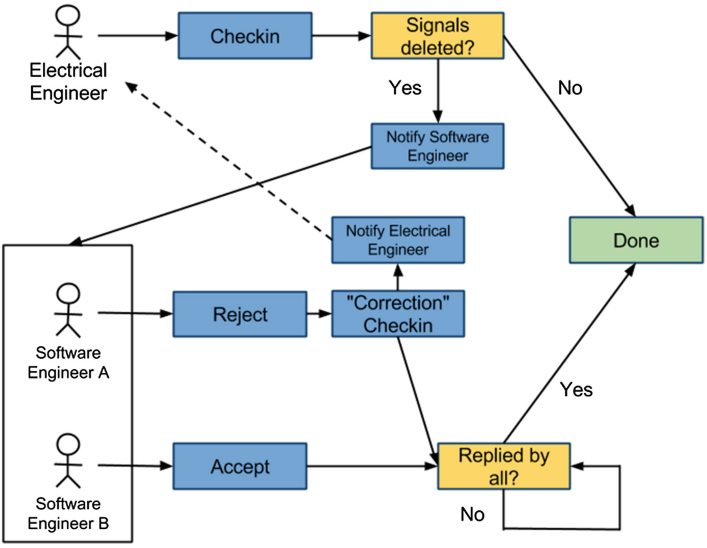

Fig. 2. Review circle of change management.

affect the local data model only), nothing else happens. Otherwise, SE-A and SE-B (let's assume there are two SE in the scenario), as being from a different discipline then EE, receive a notification about the changes which they either accept or deny.

Assuming that SE-A rejects the change, he provides corrections and publishes them. This means that after applying the (undesired) changes locally, SE-A introduces the (now) missing variables again to produce a change-set fixing their local issue. This change-set can be seen by the EE as well and could potentially trigger a new review cycle. After propagating, SE-A has voted and the review will be finished once all other participants (in this case: SE-B) have submitted a vote (or a timeout has expired). SE-B then simply accepts the change (most likely aware of SE-A's adoptions) and "unlocks" the final change-set (changes of EE with fixes of SE-A) for their continuing work.

4 Research Issues

In every multi-disciplinary software and systems engineering project, project participants from different disciplines using role-specific (and often isolated) software tools need to cooperate in order to plan, design and evaluate project results and intermediate artefacts. Most of these tools were designed for a specific phase during development and therefore only provide a limited view of the artefacts engineers work on. Based on the described challenges, we derive the following research questions:

RQ1 - Modeling Data Integration Solutions: Which modelling concept is needed to allow for seamless data integration in a heterogeneous software landscape? Out of the necessity for the different engineering groups to work together, the groups agree on some common data model for specification and progress/artefact exchange. However, with a variety of proprietary tools used in this sector and the varying engineering workflows of the companies, there is not "the" one data model to cover all scenarios, even if limited to a very specific subset (e.g. the design of hydro-power stations).

RQ2 - Modifying Existing Integrations Solutions: With reoccurring changes to the data models, how can we distinguish between valid changes and erroneous modifications? From the observed diversity of tools, data models and workflows, it is assumed that data models must cover the following aspects [29]:

- Tool Data Models (TDMs): covering proprietary models of the tools in place as best as possible
- Tool Domain Data Models (TDDMs): abstracting from the tools how engineers of one discipline view their artifacts
- Common Data Models (CDMs): the view that is used between disciplines to collaborate, and that may be addressed directly

Data models used for integration and the data models to integrate will change over time, even within a specific group or company. Our observations in the automation industry have shown that engineers of the same discipline may have very distinct workflows and modelling approaches. This is due to the fact that people are specialized in the design and implementation of different facilities and have adopted to whatever style works best for their field of expertise. Under these circumstances, an engineer may not only introduce slight differences in the importance of parts of a data model, but also introduce new fields (or discard some) for their personal "workflows".

Based on literature review, iteratively developed and implemented research prototypes helping to find data design model patterns for a specific cross-discipline engineering workflow, and evaluation of the design for a given scenario using ATAM a guideline to validate instances/updates of the design were created.

5 Data Model Candidates

This section details the design, testing and inspection of four different architectural styles, each promoting another modelling style. We illustrate the development process of the initial candidate for a data integration style and its evaluation by prototyping and user testing. The approaches have been implemented in an Enterprise Service Bus environment [6], called Engineering Service Bus[5] (EngSB) to a varying degree for testing under real-world conditions.

[5] Downloadable under http://github.com/openengsb.

Prototype-Iteration 1: Virtual Common Data Model - The VCDM [41] is based on a LAV modelling style, and depicts a variant to integrate heterogeneous data sources and derive new functionality from these sources, i.e. for versioning and quality assurance. On the client side, each tool holds data in its own model, but upon transmitting its data instances to a central bus system, the data is transformed into a single common (global) data model in the first step. After transformation, the data is stored into a central schema-less repository which holds a change history of the data as well as the actual instances, the "Engineering Data Base" (EDB) [41].

The limitation of the approach is the lack of an actual global data schema in the storage solution (which allows for easier integration of new data models) demanding schema-enforcement in the software used to push the data into the EDB. This enforces either the use of a constant data model in the top layer of the application, interlocking software and data model across several layers or the use of a highly generic data model, which creates lots of boiler-plate code on the front-end level. In both cases, changes to the data model cause code refactoring in a multitude of places, making the code hard to maintain.

Prototype-Iteration 2: Service-Based Single Domain - Align with the classic Enterprise Service Bus [6] concept each single functionality of the EngSB is encapsulated in a service and data is transferred along the services by a workflow and turning the model explicit in a single domain. The workflow encapsulates the instructions to send data from the tools along a pipeline of services before using a e.g., the "Signal Domain" (see Sect. 3) to store the data in the EDB. Illustrated services include a "Transformer Service" that translates tool data into common data and vice versa, and an "Analyzer Service" which ensures data format validity (e.g. by inserting default values into fields that should not be empty or verifying the format of primary keys) and checks new data for well-known errors (which do not violate the data model, but cause validity issues later on in the project). The "Signal Domain" (being the common data model) is acting as an interface masking the actual storage system.

Unlike the VCDM style, the approach makes the data model explicit, allowing for easier testing and refactoring of individual components, also allowing new developers to understand the common data model much easier. In addition, by using a "service pipeline", new processing steps (for new features) can easily be added without having to change existing workflow from the user's point of view. The main weakness of this design lies in the fact that changing the data model still causes changes in all services along a workflow, as there is no central authority governing the data models inside the bus system. Although these changes are now fairly easy to perform, as every service is a smaller, less complex component then before, failure to adapt all components can yield unexpected results which are not detected at first glance.

Prototype-Iteration 3: ESB-Domains only - Introduces a central authority for data models across workflows which turns every service into a domain and the common data model into the tool's entry point to the EngSB. This results in:

- smaller components: with each service of the EngSB wrapped into a domain, much boiler-plate code for setup and data transfer is delegated to the generic domain implementation. This reduces code duplication in the services and reduces maintenance costs as services only contain their own "functional" code.
- robustness against data model changes: by transforming a domain (a generic interface with an attached modell) to an entry point via third party applications interact with the EngSB, the effects of changing tool models were limited to the tool's connector, thus causing no other changes inside the EngSB. Even if the domain data model changed, it did not affect the EngSB, as the data model of a domain is never used directly, but via generic methods unaware of the actual implementation and its specific model. The only case in which a model change affects the EngSB is if a field that was explicitly addressed for workflow rules or inside an internal domain changed.
- simpler workflows: since every services deployed into the EngSB is present as a domain, their use in workflows is standardized, meaning that configuring workflows become easier.

The downside of the approach is that the EngSB enforces the domain model as a standard onto the companies using it. In addition, by keeping the tool data models "outside the EngSB", neither the workflow nor any of the EngSBs' internal services could exploit distinct features of the connected tools or provide any functionality based around tool-specific data. Additionally, connecting all third party tools via a single domain moved the focus on the role of the tool in the engineering process. Tools are either data sources or sinks but not both and have a certain hierarchy, i.e. one tool's changes will always override another tool's changes. If all tools' input is merged together into a single domain when executing a workflow, it is hard to treat them differently and calls for solutions implemented outside the EngSB.

Prototype-Iteration 4: Multiple Domains and one Engineering Object - After reorganizing tool-model relations, for each group of tools belonging to the same engineering discipline a so called tool domain was assigned relieving the "common data domain" from holding tool-specific data. The former "data domain" any coupling to specific engineering tools, and is used for data validation and cross-discipline interactions only - and is referred as "Engineering Object". The separation changed the nature of data conversions - there is a tool-to-domain ("mapping") and domain-to-domain ("transformation"). Mappings mostly consider simple field renamings and basic string operations (such as splitting, concatenating, extracting substrings and formatting) due to the fact that a tool's data model and its corresponding domain's data model are semantically similar by design. Domain-to-domain transformations may be complex since they intend to connect heterogeneous models - the prototype makes use of ontologies to describe such transformations [29]. By introducing two conversion types, the solution enables power-users to deal with frequently changing configuration of tool-to-domain mappings (as the tool's export formats are project dependent), while integration application developers can focus on tackling the domain-to-domain transformations.

The drawback of the approach is the overall higher complexity. Data models to design (tool domains and common data models) are more complex then before. Thus, it may pose an over-engineered solution if one or two tools from a single discipline should be integrated. Additionally, considering the components used for mapping, transforming, storing and processing (workflows) the data, this approach contains a higher number of isolated subsystems which are configured differently posing consequently a higher chance of "transitive" errors. A missing or invalid configuration in one component, which has no immediate negative effects, causes unintuitive behaviour in another component.

Table 1 summarizes the features of the data model design styles presented before. The table indicates that the "Multiple Domains and one Engineering Object"-style is our best candidate for a good integration solution.

Table 1. Feature comparison of the data model design styles.

	VCDM	ESB-like	ESB-Domains	EngObj
Robust against changes	no	no	yes	yes
Can hold multiple common models	no	yes	yes	yes
Versioning	yes	yes	no	yes
Service-Reuse	yes	no	no	yes
Only valid data	no	yes	yes	yes
Data sovereignty	yes	yes	no	yes

6 Evaluation

This chapter illustrates the investigation of a Engineering Object-based data modelling style by evaluating it against our main scenario (see Sect. 3) using the ATAM. Changes to the given scenario that influence mentioned data model design are listed to understand the volatility of (seemingly) stable data models.

6.1 Adapted ATAM

The execution of the ATAM (see Fig. 1) was modified to reduce the time effort for participants from industry partners and to make use of existing documentation. However, as the project chosen for this evaluation was already running, the time-frame of the ATAM was adapted from the (recommended) block of several days to a period of two months.

1. **Present the ATAM:** The process was presented in a meeting with stake-holders from industry and academia to inform project participants about the conducted ATAM process.

2. **Present business drivers:** existing minutes of meetings and knowledge of the project were used to identify the business goals and non-functional requirements. Additionally, the focus of this inspection was shifted even more in favour of non-functional requirements, as the functional requirements.
3. **Present architecture:** the architectural review is limited to the data model design style and therefore only the goals that can be covered by it.
4. **Identify architectural approaches:** Step 4 was merged into step 3, as the chosen architecture is known at this point and the relation to the business goals is being identified in step 3.
5. **Generate quality attribute utility tree:** Step 5 is left unchanged regarding its way of execution, but limited onto the data model design, in contrast to the entire architecture. This, combined with the results from step 4, allows to identify which of the business goals should be addressed via the data model design and which actually are covered this way.
6. **Analyse architectural approaches:** Step 6 is left unchanged, since it is based on results from step 5.
7. **Brainstorm and prioritize scenarios:** This step is separated into two parts. First, use cases/scenarios already covered by the prototype are evaluated in terms of user satisfaction and future (desired) use cases are voted for. Second, the results of that vote are used for clarifying existing specifications. The results and main (intermediary) results gathered and processed in this step remains the same, however the time-frame of its execution is altered again.
8. **Analyze architectural approaches:** Step 8 is left unchanged, as architecture has already been analysed by the relevant stakeholders in step 6.
9. **Present results:** The findings of the (modified) ATAM execution were presented to industry partners.

6.2 Results of the ATAM

Using the modified ATAM, the eight steps were performed to gain a better understanding of the integration project's goals and the effectiveness of the proposed architecture.

Step 2 - Present Business Drivers. From meetings three forces towards a better solution were identified: (1) Failure to properly track changes and their propagation, causing misconceptions about a project's status, (2) High costs to re-use assets from previous projects (i.e. mostly partial designs for components), and (3) Strict coupling of the inter-disciplinary workflows to each software tool in use, restricting the flexibility of all groups.

Thus, the main functional requirements to the integration solution are as follows: (1) Convert and transfer data instances from one of the tools into the respective other two models and vice versa, (2) Track changes of one's work compared to the last known working copy and changes which were propagated to other disciplines, and (3) Keep older copies of the data available to revert to those should current changes cause dissent between the engineering groups.

Step 3 and 4 - Present Architecture and Identify Architectural Approaches. The architecture / data model under review consists of five components:

Tool Data Models (TDM) which are pre-defined data models originating from the tools to integrate.

Tool Domain Data Models (TDDM) which abstract the TDMs and represent a common but tool independent model for the integration solution.

Common Data Models ("Engineering Objects") that are used to connect TDDMs with each other and allow for a generic, discipline-independent view of the data instances.

Mappings define "parsing configurations" thus enabling an import and export of "raw" tool data into their respective domains.

Transformations are the translation instructions to convert data from one TDDM or Engineering Object into another.

From the previous steps six goals the architecture has to server were identified:

- "Update Propagation & Notification" demands that specific data instances can be identified across model boundaries.
- "Ease of Refactorings" requires the ability to extract and batch-modify large sets of data.
- "Reducing Tool Restrictions" requires to check whether shortcomings of the tools' data models are isolated and bypassed by the data model design or introduced into the integration solution.
- "Data Exchange" requires that (changing) data models can hold and exchange (tool) data instances.
- "Traceability" requires identification of data instances and operations.
- "Versioning" as a general property all data storages in the integration solution need to have.

Regarding the design decisions made for the data model design, two main (modelling) concepts can be identified *Inheritance* (as TDMs can be referred to as subclasses of TDDMs) and *Abstraction* (as Common Data Models aim to aggregate "interesting" parts of the TDDMs). On the other hand, "Global-Local-As-View" can be identified as the dominant paradigm for the overall data modelling strategy, since the "main" models present TDDMS and Common Data Models treat each other as views of them, while "real" data models are used to access and modify the data instances.

Step 5 - Generate Quality Attribute Utility Tree. Although the goals "Update Propagation & Notification" and "Traceability" are based on different use cases/scenarios, they require the same quality, namely tracking the links between data instances across tool (data model) boundaries. Additionally, not all of the identified goals have the same degree of immediacy attached, as "Ease of Refactorings" is to be considered in follow-up (engineering) projects once data is already stored and handled with the application integration solution.

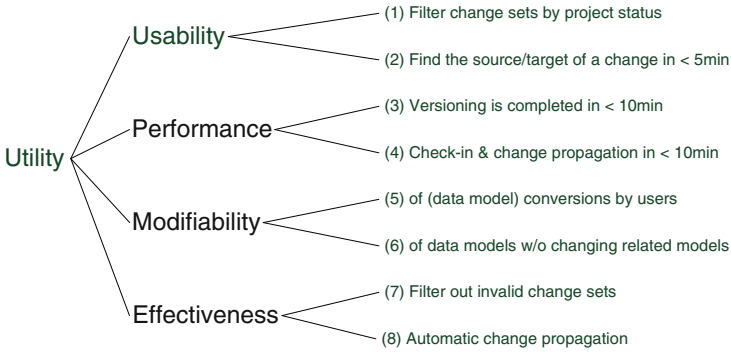

Fig. 3. Categorization of quality attributes these attributes

This reduces to four goals that are connected to several quality requirement attributes.

1. "Traceability" goes hand in hand with usability, as users may need to (re)view data change-sets and effectiveness, as invalid data instances should be filtered out.
2. "Data Exchange" is connected to performance as well as effectiveness, since a functioning data exchange that does not slow down the engineers' workflows is the core motivation to build an application integration solution.
3. "Reducing Tool Restrictions" is linked to modifiability and extensibility, as the data models "inside" the application integration solution have to embrace new features and should withstand a change of the underlying tool (or its data model).
4. "Versioning" also is linked to performance as it should not slow down "main" user interactions.

As shown in Fig. 3 eight quality attributes were derived and specified in more detail as follows: **(1)** Being able to filter important/unimportant change sets by project status. **(2)** Be able to navigate/display the source/target for a local/propagated change set in <5 min. **(3)** Versioning of change sets must be completed in <10 min (should not stall later check-ins). **(4)** A check-in and the following propagation of data has to be finished in <10 min (after user interactions). **(5)** Allow for modification of the conversions between data models by (power) users. **(6)** Allow for modification of data models without the need to change related models. **(7)** Automatically filter out invalid change sets before further propagation. **(8)** Automatically propagate changes across tool boundaries (after a check-in of data into one model).

From reviewing previous minutes of meeting and feedback from delivered prototypes, the assumed priorities (importance for the overall success) and risks (cost to satisfy) of the goals are summarized in Table 2 (sorted by priority, followed by effort):

Table 2. Priority and effort of goals.

#	Name	Priority	Effort
8	Change propagation	high	high
5	Modify conversions	high	medium
6	Modify data models	high	low
7	Filter invalid data	medium	high
1	Filter by status	medium	low
2	Change source/target lookup	low	medium
3	Fast versioning	low	low
4	Fast check-in & Follow-up	low	low

Step 6 - Analyze Architectural Approaches. There is a total of three high-priority goals which have to be analysed in further detail in order to answer the following questions:

1. Does the proposed architecture satisfy the requirements imposed by the goal?
2. Are there any risks/conditions under which the requirement will not be satisfied?
3. Does satisfying the requirements impose any trade-offs on the architecture or the final solution which may hinder addressing other goals?

Therefore, for the three high priority goals ("Automatic Change Propagation", "Modifying Data Model Conversions", "Modifying Data Models") quality attribute characterizations (QACs) were created (see [35] for details), concluding that all major business goals (regarding data integration) are addressed by the data model design.

Step 7 - Brainstorm and Prioritize Scenarios. During presentation of and discussion about the importance of the eight goals to our industry partners, they decided to change two prioritizations:

1. "Modifying Data Models without having to update related models" (6) was down-voted to low priority under the assumption that tools or their respective models do not change very often, especially not during projects. However, after showing counter-examples from the initial prototyping iterations, it was agreed that this goal should have a priority of at least medium (priority of high was not necessary as it was not immediately needed for a working prototype).
2. "Being able to filter important/unimportant change sets by project status" (1) was up-voted to high priority, as tracking and preventing changes to "finalized" data sets (i.e. "Accepted by the customer") is perceived to be a major cost-cutting factor - which was unknown until that discussion.

In addition, a new goal of medium priority was introduced with a discipline-specific scenario, called "Creating and querying custom data hierarchies". From the UI design point of view, this refers to the user being able to categorize, sort and query data instances from their discipline by more then one "hierarchies".

Step 8 - Analyze Architectural Approaches. Since the prioritization of business goals was changed, an additional QAC was created (see [35] for details). The main trade-off of the proposed data model design style is a higher complexity in comparison to a 1-or 2-tiered approach, but at the benefit of serving the major business goals.

Step 9 - Present Results. Summarizing the results from the previous steps, the validation of the data model design style using the ATAM yields the following results:

1. Six main business drivers were identified in the beginning of the ATAM execution, of which five are (partially) relevant when validating the data model design style. Out of these five goals, two are identical from the data modelling point of view.
2. These four business drivers were used to extrapolate a total of eight architecture goals, marking three as highly important for the success of the project.
3. The three quality attribute characterizations created from these goals supported the theory that the data model design style serves to satisfy the requirements.
4. "Customer" review caused a down-vote of one of the QACs, up-voted one goal to high priority and introduced a new medium-priority goal.
5. Repeated QAC creation again resulted in a positive evaluation of the proposed data model design style.

7 Discussion and Conclusion

When designing and implementing application integration solutions that enable the exchange of engineering artefacts across discipline and tool boundaries, the data model design at the very start can ultimately decide the success of such an integration effort. Most of these integration scenarios can be simplified to a fixed number of tools (usually 3 to 5) which are each used by one discipline (such as electrical, mechanical and software engineering).

By iterative prototyping of solution candidates different modelling styles were tested over the course of several months. During each iteration, a small set of use cases (2–4) regarding tool integration was taken for testing the current development iteration. Using the results from previous iterations, more detailed specifications of those use cases were created, combined with sample data to "simulate" runs of said use cases, and used to redesign current data models and architecture, and modify or extend business logic.

These architectural changes also affected the way in which tool data and common data were stored, transformed and treated, introducing a set of different data modeling styles for ostensibly similar scenarios. Styles align with the classic Enterprise Service Bus [6] and Virtual Common Data Model [41] proved to be feasible long-term solutions for the immediate purpose of model integration and tool chain support. However, through regular user feedback and iterative evaluation of the project's requirements over time, it turned out that changes to the common data model happen more frequently then expected, causing redeployment of a large number of components and intense re-configurations.

The proposed 3-layered approach, based on the Engineering Service Bus (EngSB) framework, was designed after reworking the design behind previously deployed, working prototypes which already allowed for a data exchange across 3 disciplines. This model was again evaluated using the ATAM before using it to build a specific data model design for our test scenario. A limitation of the proposed data modelling style is its relatively narrow area of applicability. The data model design style was designed for middle-sized to large engineering project with two or more (engineering) disciplines collaborating which each other. The basic assumption is that there exists at least one common data model which may be used to exchange the artefacts created by each respective group.

It may be therefore concluded that the proposed solution introduces high complexity. The 3-layered approach consists of tool data models (TDM), tool domain data models (TDDM) and common data models ("engineering objects"), combined with transformations and mappings. This implies that these three data models and two conversion configurations have to be carefully modelled and maintained. Consequently, conversations and discussions between a representative of each engineering discipline is needed to ensure the quality of the overall data model design.

As future work it is intended to minimize complexity of the proposed approach by introducing additional tool-support for model and conversation management, and reviewing its limitations and benefits in other engineering areas, like Automotive or Aerospace.

Acknowledgments. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria.

References

1. Adelman, S., Moss, L., Abai, M.: Data Strategy. Addison-Wesley Professional, Indianapolis (2005)
2. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. In: Proceedings of the 25th International Conference on Software Engineering, ICSE '03, pp. 187–197. IEEE Computer Society, Washington, DC (2003)
3. Biffl, S., Schatten, A., Zoitl, A.: Integration of heterogeneous engineering environments for the automation systems lifecycle. In: 7th IEEE International Conference on Industrial Informatics, INDIN 2009, pp. 576–581 (2009)

4. Biffl, S., Schatten, A.: A platform for service-oriented integration of software engineering environments. In: Proceedings of the 2009 Conference on New Trends in Software Methodologies. Tools and Techniques: Proceedings of the Eighth SoMeT'09, pp. 75–92. IOS Press, Amsterdam (2009)
5. Boucké, N., Weyns, D., Schelfthout, K., Holvoet, T.: Applying the ATAM to an architecture for decentralized control of a transportation system. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 180–198. Springer, Heidelberg (2006)
6. Chappell, D.: Enterprise Service Bus: Theory in Practice. O'Reilly Media, New York (2004)
7. Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding ui integration: a survey of problems, technologies, and opportunities. IEEE Internet Comput. **11**(3), 59–66 (2007)
8. Fan, H.: Investigating a Heterogeneous Data Integration Approach for Data Warehousing. Ph.D. Thesis, School of Computer Science & Information Systems Birkbeck College (2005)
9. Fay, A., Biffl, S., Winkler, D., Drath, R., Barth, M.: A method to evaluate the openness of automation tools for increased interoperability. In: Industrial Electronics Society, IECON 2013–39th Annual Conference of the IEEE, pp. 6844–6849, Nov 2013
10. Ferber, S., Heidl, P., Lutz, P.: Reviewing product line architectures: experience report of ATAM in an automotive context. In: van der Linden, F.J. (ed.) PFE 2002. LNCS, vol. 2290, p. 364. Springer, Heidelberg (2002)
11. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston (1999)
12. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Comput. Surv. **28**(2), 415–435 (1996)
13. Giordano, A.: Data Integration Blueprint and Modeling: Techniques for a Scalable and Sustainable Architecture. IBM Press, Pearson (2011)
14. Gritton, B.: Inter-enterprise integration x2014; moving beyond data level integration. In: OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges, pp. 1–10 (2009)
15. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: the teenage years. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB '06, pp. 9–16. VLDB Endowment (2006)
16. Hentrich, C., Zdun, U.: Patterns for business object model integration in process-driven and service-oriented architectures. In: Proceedings of the 2006 Conference on Pattern Languages of Programs, PLoP '06, pp. 23:1–23:14. ACM, New York (2006)
17. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional, Boston (2003)
18. IBM Coporation: How service-oriented architecture (soa) impacts your it infrastructure (2011-2008)
19. Islam, S., Rokonuzzaman, M.: Adaptation of atamsm to software architectural design practices for organically growing small software companies. In: 12th International Conference on Computers and Information Technology, ICCIT '09, pp. 488–493 (2009)
20. Kamina, T., Tamai, T.: Lightweight scalable components. In: Proceedings of the 6th International Conference on Generative Programming and Component Engineering, GPCE '07, pp. 145–154. ACM, New York (2007)

21. Kazman, R., Klein, M., Clements, P.: Atam: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Carnegie Mellon Uiversity, Software Engineering Institute (2000)
22. Lenzerini, M.: Data integration: a theoretical perspective. In: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, pp. 233–246. ACM, New York (2002)
23. Linthicum, D.S.: Enterprise Application Integration. Addison-Wesley Professional, Reading (1999)
24. McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: 19th International Conference on Data Engineering, 2003, Proceedings, pp. 227–238 (2003)
25. Meyer, B.: Reusability: the case for object-oriented design. IEEE Softw. **4**(2), 50–64 (1987)
26. Microsoft Corporation: Integration Patterns (Patterns & Practices). Microsoft Press (2004)
27. Mili, H., Mili, F., Mili, A.: Reusing software: issues and research directions. IEEE Trans. Softw. Eng. **21**(6), 528–562 (1995)
28. Kwakye, M.M., Kiringa, I., Viktor, H.L.: Merging multidimensional data models: a practical approach for schema and data instances. In: DBKDA 2013, The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications, pp. 100–107 (2013)
29. Moser, T., Biffl, S.: Semantic integration of software and systems engineering environments. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **42**(1), 38–50 (2012)
30. Paulheim, H.: Ontology-Based Application Integration. Springer, Berlin (2011)
31. Pena, C., Bastarrica, M.C., Perovich, D.: Atam-hw: extending atam for explicitly incorporating hardware-related trade-off decisions. In: Proceedings of the 2010 XXIX International Conference of the Chilean Computer Science Society, SCCC '10, pp. 119–123. IEEE Computer Society, Washington, DC (2010)
32. Raza, A., Abbas, H., Yngstrom, L., Hemani, A.: Security characterization for evaluation of software architectures using atam. In: International Conference on Information and Communication Technologies, ICICT '09, pp. 241–246 (2009)
33. Reeve, A.: Managing Data in Motion: Data Integration Best Practice Techniques and Technologies (The Morgan Kaufmann Series on Business Intelligence). Morgan Kaufmann, Burlington (2013)
34. Reijonen, V., Koskinen, J., Haikala, I.: Experiences from scenario-based architecture evaluations with ATAM. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 214–229. Springer, Heidelberg (2010)
35. Mordinyi, R., Winkler, D.: F.W.S.B.: Ifs-cdl-14-15 - integrating heterogeneous engineering tools and data models: A roadmap for developing architecture variants. Technical report, Vienna University of Technology (2014)
36. Schafer, W., Wehrheim, H.: The challenges of building advanced mechatronic systems. In: 2007 Future of Software Engineering, FOSE '07, pp. 72–84. IEEE Computer Society, Washington, DC (2007)
37. Schwinn, A., Schelp, J.: Design patterns for data integration. J. Enterp. Inf. Manag. **18**(4), 471–482 (2005)
38. Selby, R.: Enabling reuse-based software development of large-scale systems. IEEE Trans. Softw. Eng. **31**(6), 495–510 (2005)
39. Smith, S.: http://programmer.97things.oreilly.com/wiki/index.php/don't_repeat_yourself. Accessed 2 July 2013

40. van der Storm, T.: Generic feature-based software composition. In: Lumpe, M., Vanderperren, W. (eds.) SC 2007. LNCS, vol. 4829, pp. 66–80. Springer, Heidelberg (2007)
41. Waltersdorfer, F., Moser, T., Zoitl, A., Biffl, S.: Version management and conflict detection across heterogeneous engineering data models. In: 2010 8th IEEE International Conference on Industrial Informatics (INDIN), pp. 928–935 (2010)
42. Zhong, F.: Geological data integration and sharing on the semantic level. In: 2012 Fourth International Conference on Computational and Information Sciences (ICCIS), pp. 369–372 (2012)

Evaluation of JavaScript Quality Issues and Solutions for Enterprise Application Development

André Nitze(✉)

Berlin School of Economics and Law, 10825 Berlin, Germany
andre.nitze@hwr-berlin.de

Abstract. Today's web applications heavily employ JavaScript to cover wide parts of the applications' front ends. The scripting language, often called the "lingua franca" of the web, is also becoming increasingly popular for building enterprise-grade applications. However, in practice, the language is criticized for being too unstructured, flawed and inadequate for such projects. In this contribution, the major problems with JavaScript as the language of choice for enterprise applications are analyzed and possible solutions to ensure the quality of such systems are discussed.

Keywords: JavaScript · Enterprise applications · Software quality · Maintainability · Design patterns

1 Introduction

JavaScript is a functional, prototype-based and object-oriented scripting language. Since its hasty and politically biased creation in 1995 [26], Web developers were using it to make web sites more dynamic and user-friendly. Average scripts were well under 150 lines of code. With the introduction of the "jQuery" library in 2006 and its extensive dissemination, JavaScript could be used to enhance web sites' functionality and usability due to easy manipulation of the document object model (DOM) and the ability of asynchronous remote calls. The language is interpreted by all relevant browsers implementing the same ECMAScript standard from 2009 (ECMA-262 Edition 5.1, [7]). The responsiveness, functionality and decreased loading times which can be achieved by client-side scripting are crucial to meet the high expectations of users and provide a contemporary user experience.

There also has been a significant increase in use of web technologies (Java Script, HTML and CSS) to deliver more complex web-based and mobile software. But regarding the history of the language, it seems justified to challenge the adequacy of the language for larger projects and its ability to address the needs of enterprise software development. There is no doubt that full-stack JavaScript applications are feasible and can be highly functional and fast. The question is

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 108–119, 2015.
DOI: 10.1007/978-3-319-13251-8_7

rather: Is it economically worthwhile to produce and maintain complex and long-term enterprise software products with JavaScript today? And if so, how can the common assumptions and objections about JavaScript as a "toy language" [19] be addressed? This is why, in this paper, these issues shall be evaluated and contrasted with possible solutions in the context of software maintainability.

2 JavaScript in the Enterprise

Although the choice of the programming language should be an implementation detail [29], the decision for or against specific technologies for large organizations should rely on the technologies' aptness to the use case and its estimated sustainability. Sustainability includes the languages' prospective market share, its advancement and third-party support, the availability of developers and tools and its long-term maintainability.

2.1 Enterprise Software

Software development in large organizations is distinct from the approach of developing software with few developers and limited scope. Although there is no generally accepted definition of "Enterprise Software" [12], the discussion can be compared to the distinction between "tinkering" and "engineering" [10]. While badly written, smaller software systems can provide their services for many years without significant side effects, enterprise software tends to show earlier warning signs like performance degradation, hard extensibility and increased maintenance effort as business needs change. This is why these products need a solid development approach and a structural foundation which average developers can understand and extend.

There are several models for defining and measuring the quality of a software product. The ISO/IEC 25010 quality model (Systems and software Quality Requirements and Evaluation) [15], e. g., defines eight quality characteristics: *Functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability.*

Some characteristics gain in importance as the software matures. One of the core concerns for enterprise applications is the maintainability of the software as these applications are supposed to be used for more than a few years and usually for more use cases than they were originally intended for.

Maintainability can be composed of different quality criteria, e. g., understandability, modularity, changeability, extensibility and testability. It is generally accepted that maintainability is heavily influenced by the structural quality of the software product. The metrics of McCabe [18] and Halstead [14] have been popularized beside others to measure the quality of code. Also, there are several static code analysis tools which indicate problematic constructs and actual errors. On the architectural level there are mainly manual assessments like architecture reviews or the architecture trade-off analysis proposed by Kazman et al. [16]. To evaluate the overall suitability of JavaScript in terms of maintainability,

low-level code quality aspects must be discussed along with higher-level ideas like modularity.

2.2 Related Work

Much research in terms of JavaScript has been done on obfuscation techniques. However, researchers have not yet explored the role of JavaScript in the enterprise and its implications on software quality and especially maintainability.

Mikkonen and Taivalsaari, despite being in favor of JavaScript, expressed several concerns [19]. Among others, they comment on the extreme permissiveness of the language, the maturity of core libraries, the lack of modularity and classes, syntactic issues, an "overly complex and clumsy to use" I/O model, lack of an "include" mechanism and performance and memory management issues.

A representative analysis of the runtime behaviour of real-world JavaScript programs was conducted to verify several assumptions about the language [25]. The analysis revealed malpractices like the frequent and defective use of `eval`, change of types even late in an objects' lifespan, change of built-in object types and redundant code. Furthermore, several typical JavaScript code smells have been identified by Fard and Mesbah [11].

Ocariza et al. conducted empirical studies of client-side JavaScript code and found many DOM-related bugs [22] and errors in popular production web applications [23]. They also found that many of these errors (72 %) are non-deterministic, i. e., they vary across executions.

In previous research, the modularity aspect of JavaScript frameworks and libraries has been discussed in more detail [20].

3 Problems with JavaScript Software Development

The origin of some of the problems stated above and expressed by practitioners [1–3] can be seen in the backwards compatibility needed to support older browsers and web sites. Others may be rooted in the original intention of the language as a "lightweight complement" to Java [26]. Regarding the history of the language, its suitability for large software projects has to be questioned. Therefore, typical examples of the most often cited inadequacies shall be presented here.

3.1 Structure

The application structure can be organized via modules. A module in terms of web applications can be considered a functionally self-contained part of the application. Communication between modules should be conducted using a mediator. The widely known principles behind modularity are loose coupling and high cohesion. It is well-known, that high modularity of software significantly increases its quality characteristics, especially testability and extensibility.

In JavaScript, there is no `import` or `include` directive known from other languages, which could help to structure the application. This is a common problem when application modules are to be loaded dynamically at run time. It occurs, when there are variable collisions in the global scope, e. g., two variables have the same name and the first one is overridden by the other. In JavaScript, this can happen easily due to *implicit global variables* and the use of variables without declaration. Namespaces are the corresponding design pattern to solve this problem. But namespaces are not part of the standard and thus are typically provided by additional "module loaders".

3.2 Typing

The dynamic typing of JavaScript is convenient at first but it can have severe consequences. For example, having "flexible" types prevents static analysis tools from finding problems in type conversions, interface definitions and comparisons. The "duck typing" approach – determining the type of an object during runtime and according to its context at a given point of time – is very flexible but can be counterproductive in a corporate environment, where lots of differently skilled people have to read and extend source code. – In particular, when that code does not execute with easily predictable and comprehensible results. In that case, it could be argued, that the additional code of statically typed languages would add to the understandability of the program. A static type system, on the other hand, can only prevent basic type errors as there are no semantic checks. Despite the general discussion around type systems, an enterprise-grade language will most likely be required to provide some guarantee on types to be accepted by managers in terms of risk prevention.

3.3 Inconsistencies

There are inconsistencies in the language, which make it easy to create unexpected program behavior. This holds especially true for programmers who are used to languages which had the chance to be created for and matured in enterprise environments.

Special language constructs. In JavaScript, everything is an object. A problem with this is, that even *null* is an object, which can result in obscure behavior at comparisons.

`eval()` and `with` are two other heavily debated language constructs. `eval` enables the interpretation of strings as native code and thus can convey serious security issues (Sect. 4.1). The comfortable `with` also has more harmful than appropriate uses, which contributes to its use being a bad practice[1].

Private properties of JavaScript "objects" can be defined within closures. Public variables and methods can be exposed using different scopes and the

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/with

`return` statement enabling the prototypes to provide interfaces. However, this way of providing object-oriented features is neither intuitive nor fail-safe.

Scope Ambiguity. Scopes normally help to reduce naming collisions and provide automatic memory management. In JavaScript, however, scopes sometimes create additional problems, as it can be seen in the following listing [31]:

```
function foo(fn) {
    if (typeof fn ==="function") {
        fn();
    }
}
var bar = {
    barbar : "Hello, World!",
    method  : function() {
        alert(this.barbar);
    }
};
bar.method();                       // alerts Hello, World!
foo(bar.method);                    // alerts "undefined."
foo(function() { bar.method(); }); // alerts Hello, World!
```

In the example, a function `foo` is created, which calls a function with the name equal to the parameter `fn`. Then, an object `bar` is created with a property and a method to alert the property. The call of the following expression `foo(bar.method)` evaluates to "undefined". The desired output can be achieved using the more complex and less legible `foo(function()  bar.method();)`. These inconsistencies are problematic as they cause defects which are hard to trace and thus introduce significant risk into projects.

No Classes. As there are no classes in JavaScript, objects are created using a constructor function:

```
function MyObject() {
    this.myString ="Hello World!";
    this.myInteger = 0;
}
myObject1 = new myObject();
```

JavaScript is a Prototype-based object-oriented language. The prototype is a creational pattern [13, p.10] which allows the developer to use objects as templates for other objects. The prototype object is cloned and can then be extended. This can result in complexity as new subclasses can be created at runtime and the inheritance hierarchy can grow unrestricted. Additionally, the cloning process can be very expensive.

Use of `this`. The use of `this` is problematic when it is used inside of methods. The object-oriented programmer expects to operate on an object when using `this`. In JavaScript however, the use of `this` depends on the scope it is used in. Consider the following code listing (adapted from [4]):

```
function MyObjectConstructor(message){
    this.message = message;
    this.show = function(){
        alert(this.message);
    };
}
myObject1 = new MyObjectConstructor("my test string");
var methodpointer = myObject1.show;
methodpointer();        // alerts 'undefined'
```

In this case, `methodpointer()` should access the method `show` of the object instance `myObject1` to alert the `message` property. But instead, it looks for `this` in the global scope. Thus, the execution of the above script results in an "undefined"-error. There are workarounds to solve this problem (i. e., using an intermediate "that" variable, s. [27, p.47]), but it is not the behavior expected by developers. Another example, derived from Crockford [9, p.28f], again shows unexpected behavior regarding the scope:

```
var value = 0;                // outer value (global context)
var myObject = {
    value : 1,                // inner value (object context)
    increment : function () {
        var helper = function () {
            this.value++;     // references outer value (wrong)
        };
    helper();
    }
};
document.writeln(myObject.value); // 1 (correct)
myObject.increment();             // invoke increment method
document.writeln(myObject.value); // 1 (wrong, should be 2)
document.writeln(value);          // 1 (wrong, should be 0)
```

Internal Code Completion. In the case of JavaScript, even code style has an impact on functionality as commas and semicolons are added automatically. This convenient function can lead to weird behavior as can be seen in the following listing [31]:

```
return
{
  foo : bar
};
```

For the interpreter, the code looks like this after the "improvement" of semicolon insertion, ignoring the intended object creation:

```
return;      // JavaScript incorrectly adds this semicolon
{
    a :'b'; // semicolon added
};
```

Global Variables. Many additional libraries and frameworks add variables to the global scope to make their functionality available to other modules. The problem is, that these variables "pollute" the global namespace, possibly resulting not only in lack of overview but also in conflicts with other variables which have been defined a priori.

3.4 Scalability

The extensive use of JavaScript as a front end technology constitutes a paradigm shift to a more client-side focus of the application architecture with all the inherent benefits and drawbacks. A major benefit is, that the scaling of an application is less of an issue as significant parts of the main workload are processed on the client. Scaling can be done on the server side by employing an event-driven architecture, which fits well to the non-threaded nature of JavaScript. This, however, can require significant changes within the enterprise architecture and thus can be a severe project constraint.

4 Possible Solutions

There are several directions from which the above mentioned problems can be resolved. The solutions differ in terms of radicality, e. g., from relatively unobtrusive programming standards to changing to completely other primary languages.

4.1 Standards

Making use of programming standards or conventions should be common in enterprises as they pave the way to decent software quality. These standards range from high-level *principles* (e. g., Separation of Concerns, Don't Repeat Yourself, Keep It Short and Simple) to more operational *practices* (e. g., don't mix code and data; comment your code; apply naming conventions; don't change objects you don't own) and very basic *conventions* (e. g., indentation with tabs; no null comparisons; camel case and verbs for methods).

The application of architectural design patterns ensures long-term maintainability and extensibility of the software. Style guides, in turn, can be less obstructive when they are automatically applied when pushing code to a version control system or during the build process.

There are several resources on best practices and coding standards in JavaScript:

- Google JavaScript Style Guide[2]
- jQuery JavaScript Style Guide[3]

[2] http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml
[3] http://contribute.jquery.org/style-guide/js/

– Crockford/Rashbrook JavaScript Code Conventions[4]
– Dojo Style Guide[5]

There are integrated tools to automatically check for these (s. [8,27]). Practitioners and scholars also have collected generic and JavaScript-specific programming best practices (s. [9,13,17,24,27,33]). With such standards, testable and legible JavaScript code can be produced and checked at least partly automated while avoiding common programming pitfalls.

4.2 Language Evolution

As of writing this paper, the most recent version of the ECMAScript standard is 5.1 [7]. The draft of the next version ("Harmony"[6]) addresses many of the stated issues: `module`, `import` and `export` declarations, object literals, static typing and arrow functions. "Troublesome" constructs are to be removed while preserving "the start small and iteratively prototype nature of the language".

The draft documentation also names the goals of the new version (excerpt):

– Be a better language for writing:
  – complex applications;
  – libraries (possibly including the DOM) shared by those applications;
  – code generators targeting the new edition.
– Switch to a testable specification [...].
– Improve interoperation, adopting de facto standards where possible.

This shows, that the language is being changed and extended to better support the desired use cases of more complex, i. e., enterprise applications, but also to prevent the issues stated above. However, it remains unclear, when the new version will be completed, released and implemented by browser vendors. A problem, which could arise from this process, is a fragmented support of different JavaScript versions by web browsers.

4.3 Intermediary Languages

The main criticism against the standard is its time-to-market and the amount of usable features probably included. An intermediary language could be used to compensate the lack of some features. A survey from 2013 showed, that 22 % of the participating developers used an intermediary language, most of them (85 %) CoffeeScript [28].

TypeScript is another option and an early interpretation of the new Java Script-version (ECMAScript 6). The code allows for ECMAScript-compliant code, but provides a module system with namespaces, static typing and other features which can be used immediately and without breaking old code. CoffeeScript, while being already applied in practice, only covers a subset of the capabilities of TypeScript.

[4] http://javascript.crockford.com/code.html
[5] http://dojotoolkit.org/community/styleGuide
[6] http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

The TypeScript source files are translated and compiled into ECMAScript 3-compatible files, which can be interpreted by all major web browsers. Essentially, TypeScript reduces the complexity of JavaScript syntax to write object-oriented code. The more advanced object-oriented features are used (e. g., inheritance, type casting, optional parameters), the more complex the resulting code gets. An example can be found in the following code listing [5]:

```
class BankAccount {
    balance = 0;
    deposit(credit: number) {
        this.balance += credit;
        return this.balance;
    }
}
```

The above code would be translated into this standard-compliant code:

```
var BankAccount = (function () {
    function BankAccount() {
        this.balance = 0;
    }
    BankAccount.prototype.deposit = function(credit) {
        this.balance += credit;
        return this.balance;
    };
    return BankAccount;
})();
```

But most of the documentation, best practices and conventions are based on the standard implementation. Changing this may create barriers for new developers and increase the effort to port the intermediary language back to the specification-compliant one depending on how differently they implement the distinct features.

4.4 Another Language

Considering other languages to replace or circumvent JavaScript is a justifiable option for those who want to leverage existing resources while profiting from the speed and cross-platform capability of JavaScript.

Google's "Dart" language has recently attracted developers' attention. It is positioned as a new virtual machine (VM) for the web beside and, in the long-run, instead of the JavaScript interpreter. It provides classes, namespaces and optional static types. Nevertheless, the language is not ready for production and thus cannot be considered a solution yet.

Another approach is to use JavaScript as intermediary language using cross-compilers like Emscripten [32]. That way, developers can leverage their experience in Java, C++, C# or Python to build JavaScript-based web applications. The Google Web Toolkit (GWT) is a popular and production-ready example of this approach using a Java-to-JavaScript cross-compiler. However, it is very hard to maintain or even edit the resulting JavaScript as it is highly optimized.

Table 1. Problems of JavaScript in the enterprise and possible solutions

	Structure (3.1)	Typing (3.2)	Language inconsistencies (3.3)	Scalability (3.4)
Standards (4.1)	YES	NO	YES	NO
Intermediary languages (4.2)	YES	YES	YES	NO
Language evolution (4.3)	YES	YES	YES	NO
Another language (4.4)	YES	NO	YES	NO
Ecosystem (4.5)	YES	YES	NO	NO

4.5 Ecosystem

There is a remarkable amount of open-source frameworks and libraries available for JavaScript web development. Frameworks provide application structure and solutions for typical scenarios. In particular, most web frameworks implement the separation of concerns principle using derivatives of the model-view-controller and observer patterns and provide the syntactic sugar developers are used to from other languages. Module loaders help to handle the task of resolving dependencies in large projects. *Asynchronous Module Definition* (AMD [6]) has become the de-facto standard for managing module dependencies in many frameworks. The back end platform *Node.js* provides over 70.000 packages for a vast amount of use cases at the time of writing [21]. *jQuery UI* offers hundreds of plugins which build on and extend the core library. *jQuery* itself is one of the best examples to show how the language can be used to provide great functionality without leaving the standard. In terms of design patterns the library helps to hide the complexity of cross-browser incompatibility using facades. For example, the well-known CSS selector (`$("#id")`) abstracts the browsers' different ways of selecting an element from the DOM [24].

One drawback of using a framework lies in its uncertain long-term availability. The continuity of an open-source project is a risk for enterprises. Beside the risk of "framework lock-in", there always is an overhead in terms of performance in exchange for functionality and compatibility.

The ecosystem also includes tools to scaffold, develop, build, test and deploy software products. The availability, maturity and integration of such development tools often is significantly correlated with the popularity of the language they support. Hence, these topics are considered a short-term problem, which will be solved with further dissemination of the language in professional software development companies. The TIOBE programming language index ranks JavaScript at the 9th position as of May 2014 [30] indicating a significant importance which cannot be ignored by enterprises.

5 Conclusion

In this paper, an overview of general and enterprise-specific problems with JavaScript has been given. Issues that are essential for several quality factors,

especially maintainability, have been described and illustrated with code examples. The problems discussed are compared against the analyzed solutions in Table 1. In accordance with the findings of the analysis, the question whether JavaScript is suitable for enterprise-grade applications, can be answered with "Yes". Although the language and its ecosystem are evolving, JavaScript and associated frameworks can be used to develop and maintain modern web-based applications in enterprise environments. Many of the problems with JavaScript can either be solved by using workarounds or additional frameworks and libraries. Some of the languages' problems are rooted in its specification and former use cases (language inconsistencies) while others are the result of misconceptions and prejudices. In fact, many deficiencies can be circumvented or alleviated by employing one or more of the solutions stated above. The most significant contribution to its appropriateness for large software projects is the foreseeable object-orientation, which allows for better maintainability, and hence software quality. TypeScript is considered a worthwhile intermediate language which can be used until inconsistencies are fixed in the upcoming version(s) of JavaScript.

References

1. Why javascript still sucks /r/programming (2012). http://www.reddit.com/r/programming/comments/14wk9o/why_javascript_still_sucks/
2. Java script sucks (2014). http://c2.com/cgi/wiki?JavaScriptSucks
3. Yourlanguagesucks (2014). https://wiki.theory.org/YourLanguageSucks#JavaScript_Sucks_because
4. Administrator. The this problem, 19 May 2014. http://www.i-programmer.info/programmer-puzzles/137-javascript/1922-the-this-problem.html
5. Hejlsberg, A., Lucco, S.: Typescript language specification. http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf
6. Burke, J.: amdjs/amdjs-api, 27 May 2014. https://github.com/amdjs/amdjs-api/wiki
7. Charollais, P.: Final draft standard ECMA-262 edition 5.1, March 2011. (Revised 6)
8. Crockford, D.: Jslint, the javascript code quality tool, 08 April 2014. http://www.jslint.com/
9. Crockford, D.: JavaScript - The Good Parts: Working with the Shallow Grain of JavaScript. O'Reilly, Farnham (2008)
10. Dahlbom, B., Mathiassen, L.: Computers in Context: The Philosophy and Practice of Systems Design. NCC Blackwell, Cambridge, MA (1993)
11. Fard, A.M., Mesbah, A.: JSNose: detecting javascript code smells. In: IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 116–125 (2013)
12. Foy, B.D.: What is enterprise software? 16 May 2014. http://www.perlmonks.org/?node_id=504043
13. Gamma, E.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading, Mass (1995)
14. Halstead, M.H.: Elements of Software Science. Operating and Programming Systems Series, vol. 2. Elsevier, New York (1977)

15. International Organization for Standardization. ISO/IEC 25000:2014 - systems and software engineering - systems and software quality requirements and evaluation (square) - guide to square, 19 May 2014. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=64764
16. Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J.: The architecture tradeoff analysis method. In: Fourth IEEE International Conference, pp. 68–78, 10–14 Aug 1998
17. MacCaw, A.: Developing JavaScript Web Applications. O'Reilly, Sebastopol, California (2011)
18. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. **4**, 308–320 (1976)
19. Mikkonen, T., Taivalsaari, A.: Using javascript as a real programming language (2007). http://www.activemode.com/webroot/Workers/ActiveTraining/Programming%5CJavascript_AsProgrammingLanguage.pdf
20. Nitze, A., Schmietendorf, A.: Modularity of javascript libraries and frameworks in modern web applications. In: Wuksch, D., Peischl, B., Kop, C. (eds.) Selected Topics to the User Conference on Software Quality, Test and Innovation (ASQT 2014), OCG, Klagenfurt, AT (2014)
21. Node.js. npm, 16 May 2014. https://www.npmjs.org/
22. Ocariza, F., Bajaj, K., Pattabiraman, K., Mesbah, A.: An empirical study of client-side javascript bugs. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 55–64 (2013)
23. Ocariza Jr., F.S., Pattabiraman, K., Zorn, B.: Javascript errors in the wild: an empirical study. In: IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE), pp. 100–109 (2011)
24. Osmani, A.: Learning JavaScript Design Patterns. O'Reilly Media, Sebastopol, CA (2012)
25. Richards, G., Lebresne, S., Burg, B., Vitek, J.: An analysis of the dynamic behavior of javascript programs. In: Zorn, B., Aiken, A. (eds.) The 2010 ACM SIGPLAN Conference, pp. 1–12 (2010)
26. Severance, C.: Javascript: designing a language in 10 days. Comput. **45**(2), 7–8 (2012)
27. Stefanov, S.: JavaScript Patterns. O'Reilly, Sebastopol, CA (2010)
28. Swift, J.: Javascript developer survey 2013, 23 Dec 2013. http://www.i-programmer.info/news/167-javascript/6746-javascript-developer-survey-2013.html
29. Szegedi, A.: Javascript in the enterprise. Presentation at QCon (2009)
30. TIOBE Software. Tiobe index, 05 June 2014. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
31. Way, J.: Top 10 things that javascript got wrong - tuts+ code tutorial (2010). http://code.tutsplus.com/tutorials/top-10-things-that-javascript-got-wrong-net-9266
32. Zakai, A.: Emscripten: an llvm-to-javascript compiler (2011). http://davideglintine-new.googlecode.com/hg/docs/paper.pdf
33. Zakas, N.C.: Maintainable Javascript. O'Reilly, Sebastopol, CA (2012). (Revised edition)

Refinement-Based Development of Software-Controlled Safety-Critical Active Medical Devices

Atif Mashkoor[1(✉)], Miklos Biro[1], Marton Dolgos[2], and Peter Timar[2]

[1] Software Competence Center Hagenberg GmbH, Hagenberg, Austria
{atif.mashkoor,miklos.biro}@scch.at
[2] B. Braun Medical Hungary Ltd., Budapest, Hungary
{marton.dolgos,peter.timar}@bbraun.com

Abstract. Advanced medical devices exploit the advantages of embedded software whose development, due to their direct impact on human lives, is naturally subject to compliance with the stringent requirements of safety standards and regulations. This paper presents initial results and lessons learned from an ongoing project focusing on the development of a formal model of a sub-system of a software-controlled safety-critical Active Medical Device (AMD) responsible for renal replacement therapy. The use of formal approaches for the development of AMDs is highly recommended by standards and regulations, and motivates the recent advancement of the state of the art of related methods and tools including Event-B and Rodin applied in this paper. It is expected that the presented model and analysis will contribute to the still sparse experience base available at the disposal of the scientific and practitioner community in the domain of AMDs.

Keywords: Formal methods · Event-B · Renal replacement therapy · Active medical devices · Safety-critical software

1 Introduction

The risk of chronic kidney failure increases with age, and with hypertension and diabetes becoming endemic phenomena. At the terminal phase of chronic kidney failure, renal replacement therapy is required for treatment. One of the possible forms of this therapy is hemodialysis, also known as "artificial kidney" treatment. It is a process in which using a pump system, the patient's blood is flowed through a special filter (dialyser) which filters out the accumulated waste to be removed together with the washer fluid at the other side of the dialyser. The machine responsible for this therapy is the classical example of an Active Medical Device (AMD).

The council directive 93/42/EEC of the European Union (EU) concerning medical devices [1] classifies any medical device as an AMD whose operation depends on a source of electrical energy or any source of power other than

D. Winkler et al. (Eds.): SWQD 2015, LNBIP 200, pp. 120–132, 2015.
DOI: 10.1007/978-3-319-13251-8_8

that directly generated by the human body or gravity and which acts by converting this energy. Earlier AMDs were mostly based on hardware solutions. However, lately embedded software has shown to have a determining impact on the consumer value of AMDs and their competitive differentiation. Consequently, according to the latest directive 2007/47/EC of the EU concerning medical devices [2], a stand-alone software can also be considered as an AMD. The main reason of this change is that software lend itself to adaptation to individual requirements and requirements changes clearly much better than hardware.

As AMDs become more and more software-dependent, due to the immaterial nature of software, their certification becomes a crucial issue. Certification regimes have responded to this issue by proposing various related international standards such as IEC 60601-1 [3] and IEC 62304 [4]. However, instead of containing actual recommendations for techniques, tools and methods for medical software development, these standards often encourage the use of the more general IEC 61508-3 [5] as a source for good software methods, techniques and tools.

One of the key recommendations of IEC 61508-3 is to adopt formal methods for the development of software-intensive critical systems. Their use is, in fact, "highly recommended" at higher Safety Integrity Levels (SILs). The safety integrity of a system can be defined as the probability of a safety-related system performing the required safety function under all the stated conditions within a stated period of time. Highly recommended means that if the mentioned technique or measure is not used then the rationale behind this choice has to be justified during safety planning and assessment. IEC 61508-3 further states that the confidence that can be placed in the software safety requirements specification, as a basis for safe software, depends on the rigor of the techniques by which the desirable properties of the specification have been achieved.

The main aim of this paper is to present the initial findings of an ongoing project focusing on the formal development of an AMD responsible for renal replacement therapy. The ultimate goal of the project is to harness the potential of formal methods for automatically generating software that is correct by construction. The presented formal model demonstrates an example of the way in which the requirements of the software of modern AMDs can be rigorously specified through a chain of refinements to represent them at different abstraction levels. The approach demonstrated in this paper leads to a software safety requirements specification that guarantees correctness on the addressed aspects of behavior, supports verification of the specification based on systematic analysis, avoids intrinsic specification faults, and reduces ambiguities in specification writing. As the project is still in an early stage, we have not thus far reached the automatic code generation phase.

The paper is organized as follows. Section 2 provides a brief account of the formal method we have used for modeling, the process of refinement, and the process of assessing formal correctness. Sections 3 and 4 present the Event-B model and analysis of a therapy mode of AMDs respectively. Section 5 presents

the lessons learned during this exercise. Section 6 presents some related work. The paper is concluded in Sect. 7 with some proposed future work.

2 Background

2.1 The Event-B Method

The Event-B method [6] is based on Zermelo-Fraenkel set theory with the axiom of choice. It is the successor of the B method [7] for the development of complex reactive systems. We have chosen this method for its ability to represent systems at different abstraction levels using its refinement mechanism, easy to use modeling notation and extensive tool support.

A typical Event-B model is composed of machines and contexts. Machines define the dynamic behavior of the model. They include the following:

- variables, which define the state space of a machine and can be expressed using natural numbers, integers, real numbers, boolean, sets, relations, functions or any other set-theoretical construct,
- invariants, which are used either to type variables or to constrain the state space of a machine,
- variants, which are used to define the convergence property of events, i.e., they can be triggered only for a finite number of times. They are defined using either a natural number or a finite set,
- and events, which describe state transitions. They are defined as a binary relation composed of guards and actions of an event. A guard is a predicate and all the guards together construct the domain of the corresponding relation. An action is an assignment statement to a state variable and is achieved by a generalized substitution. Combined together, all actions form the range of the corresponding relation. The actions of a particular event are executed simultaneously and non-deterministically.

Contexts define the static elements of a model. They contain carrier sets, constants, axioms and theorems. Carrier sets are used to define types. Axioms are used to constrain carrier sets and constants. Theorems define properties that are derived from axioms.

Rodin [8] is the tool that supports modeling and analysis in Event-B. Rodin is built upon the Eclipse platform and is extensible by plug-ins such as the model checking and animation plug-in ProB [9]. The main tasks that are supported by Rodin are: specification of machines and contexts, their refinement, and their consistency checking by automatically generating Proof Obligations (POs). Proofs can be discharged either automatically, with the help of third-party theorem provers, or interactively.

2.2 The Refinement Process

Refinement is the process to transform an abstract specification into a more concrete one. An Event-B model can be refined in a number of ways:

- new variables and invariants can be introduced and existing invariants can be strengthened,
- existing events can be refined to include and preserve new and existing variables and invariants respectively,
- existing events can be split into several new events, and,
- completely new events can be introduced to the model.

A machine can be refined into another machine which then contains a more detailed description of the model. A machine can see several contexts, i.e., use the constants and axioms they contain. A context can also be further refined into one or more contexts and can be seen by several machines.

2.3 Assessment of Formal Correctness

A model is considered to be formally correct when it is both verified and validated. Verification of a model is achieved when it is proved that it is free from specification errors and inconsistencies. This is usually done either through the system of POs or through model checking. A proved specification ensures that it is consistent, well-defined and its events preserve its invariants. However, proving a refinement requires to prove that concrete events maintain invariants of the abstract model, maintain abstraction invariants, and, when appropriate, decrease variants monotonically. Using model checking, we make sure that states of a model are reachable, its formulas are satisfiable and it does not contain deadlocks.

Validation of a model, on the other hand, is achieved when it is demonstrated that the model is free from requirements errors and reflects the stakeholders' wishes adequately. This can be done using several techniques, e.g., animation, reviews or walktroughs. The most common way to validate a specification in the Event-B method is to animate the specification by invoking its operation semantics to inspect its behavior. It is then examined whether the specification contains the desired functionality or not.

3 Formal Specification of the Model

The module of the AMD we have chosen to demonstrate as the case study in this paper is responsible for monitoring a critical flow at set rates from a patient's arterial access.

During the requirements modeling process, following the advice of [10] to take small refinement steps, we choose to introduce one requirement per refinement level. The static data related to requirements is modeled in contexts and the general behavior of the system is presented in machines using events. The requirements are specified as machine invariants.

In order to improve the legibility of our requirements specification, as originally proposed by [11], we have classified our axioms into three groups: technical axioms, typing axioms and property axioms. This practice helps in distilling the actual software requirements from technical expressions.

As the Event-B method lacks the explicit notion of time, we have used the timing pattern for Event-B proposed by [12]. In this technique, $\mathbb{N}$ is used to model the notion of time.

In order to conserve space, the shown refinements contain only the newly introduced information instead of the complete model.

3.1 Abstract Model

The abstract model contains the following requirement:

> The software shall monitor the critical flow in the extra-corporeal circuit and if no flow is detected for more than 120 s then the software shall stop the critical flow pump and execute an alarm signal.

We first initiate a context (`Context C0`), as shown by Fig. 1, that contains the basic data to specify this requirement. It has two sets, `criticalFlowPumpingValues` that models the state of the critical flow pumping process (`Start` or `Stop`) and `Alarms` that contains different types of alarms of the system. In addition, the context also defines a constant `noFlowMaxTime` to specify the maximum permissible time, i.e., 120 s during which, if no flow is detected, the software triggers the alarm.

```
CONTEXT
  C0
SETS
  criticalFlowPumpingValues, Alarms
CONSTANTS
  Start, Stop, noFlowMaxTime, ALM382, NULL
AXIOMS
  tec1  partition (criticalFlowPumpingValues, {Start}, {Stop})
  tec2  partition (Alarms, {ALM382}, {NULL})
  typ1 noFlowMaxTime ∈ ℕ
  pro1 noFlowMaxTime = 120
END
```

Fig. 1. Context C0

The corresponding machine `M0` specifies the aforementioned requirement as shown by `inv2` in Fig. 2. The involved variable `noFlowDetectionTime` detects the time of the last critical flow and `alarm` specifies the alarm to be triggered. An additional variable `criticalFlowPumping` is introduced to set the state of the critical flow pumping process.

In order to model the behavior of the software, following events have been introduced in the machine.

- The event `INITIALISATION` is the default event to initialize the value of newly introduced variables.

```
MACHINE
  M0
SEES
  C0
VARIABLES
  noFlowDetectionTime, alarm, criticalFlowPumping
INVARIANTS
  inv1 noFlowDetectionTime ∈ ℕ // Typing
  inv2 noFlowDetectionTime > noFlowMaxTime ⇒ alarm = ALM382
  /* If no flow is detected in 120s then the alarm should be executed */
  inv3 alarm ∈ Alarms // Typing
  inv4 criticalFlowPumping ∈ criticalFlowPumpingValues // Typing
EVENTS
  Event INITIALISATION // Initialization values
    Then
      act1 noFlowDetectionTime := 0
      act2 alarm := NULL
      act3 criticalFlowPumping := Stop
  End
  Event stopCriticalFlowPumping // Stop critical flow pumping event
    Where
      grd noFlowDetectionTime > noFlowMaxTime ∧ criticalFlowPumping = Start
    Then
      act1 alarm := ALM382
      act2 criticalFlowPumping := Stop // Stop critical flow pumping
  End
  Event startCriticalFlowPumping // Start critical flow pumping event
    Where
      grd criticalFlowPumping = Stop
    Then
      act1 criticalFlowPumping := Start
  End
  Event flowDetectionClock // The clock to simulate the time for flow detection
    Where
      grd noFlowDetectionTime < noFlowMaxTime ∧ criticalFlowPumping = Start
    Then
      act1 noFlowDetectionTime := noFlowDetectionTime + 1
  End
END
```

Fig. 2. Machine M0

- The event `stopCriticalFlowPumping` actually specifies the monitoring process of the critical flow. If no flow is detected within the specified limit of time, the action part of the event stops the critical flow pumping process and triggers the related alarm.
- The event `startCriticalFlowPumping` is trivial as it only starts the critical flow pumping when it has already stopped.
- The event `flowDetectionClock` simulates the behavior of the clock that is used to measure the time of no critical flow detection. $\mathbb{N}$ is used to represent the notion of seconds.

3.2 First Refinement

The first refinement introduces the following requirement into the model:

> If the system is not in BYPASS then the software shall monitor the critical flow in the extra-corporeal circuit and if the actual critical flow is less than 70 % of the set critical flow then the software shall execute an alarm signal.

The context `C0` is extended into `C1`, as shown by Fig. 3, which introduces three new constants into the model: `SetCriticalFlow` that is used to prescribe the critical flow, `BYPASS` that is used to set the mode of the system, and `ALM755` that is the alarm to be triggered in the current requirement.

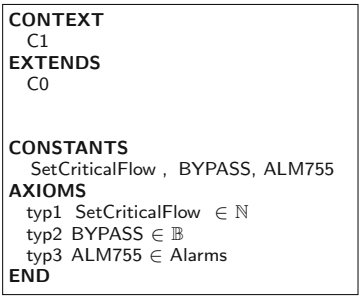

Fig. 3. Context C1

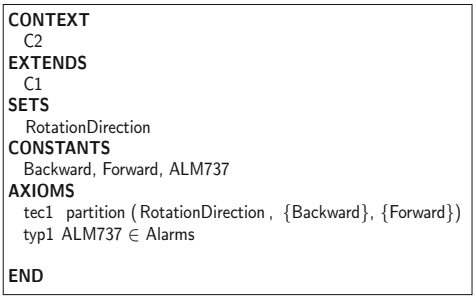

Fig. 4. Context C2

The corresponding machine `M1` at this level, as shown by Fig. 5, specifies the requirement using the invariant `inv2`. The involved variable represents the actual critical flow. The newly introduced event `checkCriticalFlow` models the behavior of the software.

3.3 Second Refinement

The second refinement includes the following requirement into the model:

> The software shall monitor the rotation direction of the critical flow pump and if the software detects that the critical flow pump rotates backwards then the software shall stop the critical flow pump and execute an alarm signal.

The context `C2`, as shown by Fig. 4, extends the context `C1` by adding a new set `RotationDirection` that models the rotation direction of the critical flow which can either be in the forward direction or backward. A new alarm type is also introduced that concerns the current requirement.

```
MACHINE
  M1
REFINES
  M0
SEES
  C1
VARIABLES
   actualCriticalFlow
INVARIANTS
  inv1  actualCriticalFlow  ∈ ℕ // Typing
  inv2 BYPASS = FALSE ∧ actualCriticalFlow < ((70/100) × SetCriticalFlow) ⇒ alarm = ALM755
  /* If the actual critical flow is less than 70% of the set critical flow
     then software shall execute an alarm signal */
EVENTS
  Event INITIALISATION // Initialization values
    Then
     act4  actualCriticalFlow  := 0
  End
  Event checkCriticalFlow // Critical flow checking event
    Where
      grd  actualCriticalFlow  < ((70/100) × SetCriticalFlow) ∧ BYPASS = FALSE
    Then
      act1 alarm := ALM755
  End
END
```

Fig. 5. Machine M1

```
MACHINE
  M2
REFINES
  M1
SEES
  C2
VARIABLES
  criticalFlowPumpRotationDirection
INVARIANTS
  inv1 criticalFlowPumpRotationDirection ∈ RotationDirection // Typing
  inv2 criticalFlowPumpRotationDirection = Backward ⇒ criticalFlowPumping = Stop ∧ alarm = ALM737
  /* If the critical flow pump rotates backwards
  THEN the software shall stop the critical flow pump and execute an alarm signal */
EVENTS
  Event INITIALISATION // Initialization values
    Then
     act5 criticalFlowPumpRotationDirection := Forward
  End
  Event rotationDetection // The event to monitor the rotation direction of critical flow
    Where
      grd criticalFlowPumpRotationDirection = Backward
    Then
      act1 criticalFlowPumping := Stop
      act2 alarm := ALM737
  End
END
```

Fig. 6. Machine M2

The corresponding machine at this level specifies the current requirement by the invariant `inv2` of the machine `M2` shown by Fig. 6. The event `rotation Direction` monitors the direction of critical flow and if it is backward, it stops the critical flow pumping process and triggers the corresponding alarm.

4 Formal Analysis of the Model

In order to verify an Event-B model, all the POs generated by Rodin must be discharged. For our model, Rodin generated three kinds of POs: (1) invariants preservation, (2) well-definedness of guards and invariants, and (3) equality of a preserved variable.

Invariants preservation relates to the condition that each variable affected by the assignment statement must preserve the invariant. For example, the event `stopCriticalFlowPumping` of machine `M0` using its guard `grd` and action `act1` ensures that `inv2` of the machine is preserved.

The notion of well-definedness relates to the condition which leads to safe evaluation of an expression. For example, `inv2` of machine `M1` states a condition where the variable `actualCriticalFlow` of type $\mathbb{N}$ is compared to an expression of type $\mathbb{R}$. However, as the value assigned to `actualCriticalFlow` is always of type $\mathbb{N}$, so well-definedness is provable.

Equality of a preserved variable amounts to proving that if a variable is present in both the abstract as well as the concrete machine and an event of the concrete machine assigns a (new) value to this variable, then it must be proven that this value is consistent with the previous one. For example, the variable `alarm` in all three machines is assigned with a new alarm, however all the alarms belong to the same type, i.e., `Alarms`.

There were a total of 19 POs. We have ensured that all of them are discharged. In fact, as the model did not contain very complex formulas, the provers were successful in discharging all of them automatically.

5 Lessons Learned

During this requirements modeling exercise, we learned the following lessons:

1. **Formal models provide a consistent and complete repository of requirements.** The information presented in this paper as requirements does not possess a one-to-one mapping from the requirements document to the requirements specification. In fact, the data related to requirements was spread across several documents. For example, the alarm numbers were not explicitly stated in the requirements. Mining the relevant data from these documents is a time-consuming and tricky task. The broken or missing links may sometimes lead to incoherent information that may impact the correctness of the model. However, one of the advantages of the current modeling exercise is also to provide a repository with adequate and consistent requirements that will positively impact the development of the software.
2. **Technical details impact the intelligibility of specifications.** The original purpose of formal specifications is to model and analyze requirements and design decisions in a way that leads to their systematic transformation into correct software. However, during the specification phase, sometimes we need to introduce additional constraints that are necessary to discharge POs but

are not part of the original requirements document. Such technical elements impede the understandability of specifications for non-technical stakeholders. The practice of classification of axioms as described in Sect. 3 not only increases the intelligibility of a specification but also helps distilling software requirements from technical constraints. The same procedure can be adopted for specifying machines. The guidelines proposed by Kossak et al. [13] for writing understandable formal specifications by using proper naming conventions and structuring also help rendering specifications intelligible.

3. **Proving versus animation.** Our requirements model is fully proven, thus we are assured that it is consistent and free from specification errors, hence correct. However, this is the modelers' point of view. From the perspective of customers, the model should also be validatable so that they can assess themselves whether their requirements are properly addressed. When we attempted to execute the model for validation purposes, we could not achieve a meaningful animation because of the high degree of abstraction at the current stage of development. As the model was simple, it was easily validated through reviews. However, during later phases of the project, when the complexity of the model increases, we expect model validation to become a crucial issue. We can, of course, lower the level of abstraction and non-determinism anytime by adding simulation scenarios to the model that would permit animation, but this approach may lead to negative effects including noise and over-specification as reported in the literature [14,15]. A middle ground has to be trodden.
4. **Model decomposition in Event-B is not straightforward.** The easy-to-use notation and the linear refinement structure of Event-B have been well-suited for modeling stringent safety requirements so far. However, as we attempt to incorporate new requirements, the size and complexity of the model becomes increasingly difficult to manage. The solution offered by Event-B to this problem is model decomposition.
   Event-B supports three types of model decomposition styles:
   - shared variable decomposition style [16] which partitions a model based on its state,
   - shared event decomposition style [17] which partitions a model based on its events, and
   - modularization style [18] which allows to introduce *interfaces* that can be made visible to other components of the model. These interfaces contain operations which can be performed by the calling module.

   The wider range of Event-B decomposition styles may be interesting for theoreticians but confusing for practitioners. Additionally, all the styles have their respective advantages and limitations (see [19] for more details). The challenges associated with these styles regarding their correctness and crude tool support (from the industrial adoption perspective), coupled with the lack of systematic guidelines and methodologies, will certainly make it necessary to engage in trial and error steps before reaching the final decision about the right decomposition style for our case.

As the requirements to be modeled are already compartmentalized into several modules, another design strategy may be to model these modules independently and then somehow compose/plug them together at the end of the modeling process. So far we have not found any such technique/methodology/plug-in for the Event-B method and Rodin platform.

6 Related Work

In recent years, the use of formal methods is escalating for the development of software-intensive medical systems. For example, Osaiweran et al. [20] use the formal Analytical Software Design (ASD) approach for developing the power control service of an interventional X-ray system, Jiant et al. [21] present a methodology based on timed automata to extract timing properties of a heart that can be used for the verification and validation of implantable cardiac devices, and Méry et al. [22,23] present an Event-B model of pacemakers. However, we could not find any literature about the application of formal methods for the modeling and analysis of AMDs for renal replacement therapy. We believe that our work will inspire the manufacturers of such systems to adopt the formal paradigm for the safe and trustworthy development of variants of this domain.

7 Conclusion and Future Work

This paper addresses the development of safety-critical software components embedded in AMDs. Ethics, as well as compliance to standards and regulations, make it imperative to follow a rigorous approach in analyzing, specifying, implementing and testing such devices. The Event-B method supported by the Eclipse-based open Rodin tool is shown to lend itself to the formal modeling of the discussed example requirements. We found Event-B an adequate method for the modeling and analysis of critical medical devices. Its easy-to-use notation, refinement principles, and verification mechanism all provide the elements that are necessary for the safe development of AMDs.

At the current refinement stage of our model, we can conclude that following the presented approach, we can continue building the model to achieve a complete and consistent requirements repository. However, it is an important consideration that, in addition to proving the verifiability of the model, it should also be validated through animation. We expect that, as the model grows and contains more concrete description of the software, it will be relatively easy to execute its behavior because of the lower level of non-determinism and abstraction.

Another future research direction is the handling of the foreseeable increasing complexity using model decomposition, which has not been fully addressed so far. We expect that this will additionally contribute towards the reusability of the model and team development. The resulting sub-models could possibly be developed by different individuals working in the same team, who can refine and manage the model in parallel in a collaborative development environment.

Once the requirements of software are sufficiently formalized, we will proceed towards their automatic translation into executable programming language code, the ultimate goal of the project. Our target language is C and there exist a couple of plug-ins, such as [24], for the Rodin platform that claim to transform the Event-B code into C language constructs. We want to assess to what extent this automatic transformation is possible in our case.

References

1. EU: Council Directive 93/42/EEC. Official Journal of the European Union, June 1993
2. EU: Directive 2007/47/EC of the European Parliament and of the Council. Official Journal of the European Union, September 2007
3. IEC 60601–1:2005: Medical electrical equipment Part 1: General requirements for basic safety and essential performance (2005)
4. IEC 62304:2006: Medical device software - Software life cycle processes (2006)
5. IEC 61508–3 Ed 2.0: Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements (2010)
6. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
7. Abrial, J.R.: The B Book. Cambridge University Press, New York (1996)
8. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T., Mehta, F., Voisin, L.: Rodin: an open toolset for modelling and reasoning in Event-B. STTT **12**(6), 447–466 (2010)
9. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003)
10. Mashkoor, A., Jacquot, J.P.: Guidelines for formal domain modeling in Event-B. In: HASE'11, pp. 138–145. IEEE (2011)
11. Mashkoor, A., Jacquot, J.P.: Domain engineering with Event-B: some lessons we learned. In: RE'10, pp. 252–261. IEEE (2010)
12. Cansell, D., Méry, D., Rehm, J.: Time constraint patterns for Event B development. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 140–154. Springer, Heidelberg (2006)
13. Kossak, F., Mashkoor, A., Geist, V., Illibauer, C.: Improving the understandability of formal specifications: an experience report. In: Salinesi, C., van de Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 184–199. Springer, Heidelberg (2014)
14. Meyer, B.: On formalism in specifications. IEEE Softw. **2**(1), 6–26 (1985)
15. Hayes, I., Jones, C.: Specifications are not (necessarily) executable. Softw. Eng. J. **4**, 330–338 (1989)
16. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: application to Event-B. Fundamenta Informaticae **77**(1–2), 1–28 (2007)
17. Butler, M.: Decomposition structures for Event-B. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 20–38. Springer, Heidelberg (2009)
18. Iliasov, A., Troubitsyna, E., Laibinis, L., Romanovsky, A., Varpaaniemi, K., Ilic, D., Latvala, T.: Supporting reuse in Event B development: modularisation approach. In: Frappier, M., Glässer, U., Khurshid, S., Laleau, R., Reeves, S. (eds.) ABZ 2010. LNCS, vol. 5977, pp. 174–188. Springer, Heidelberg (2010)

19. Hoang, T.S., Iliasov, A., Silva, R., Wei, W.: A survey on Event-B decomposition. ECEASST **46**, 1–15 (2011)
20. Osaiweran, A., Schuts, M., Hooman, J., Wesselius, J.: Incorporating formal techniques into industrial practice: an experience report. Electron. Notes Theor. Comput. Sci. **295**, 49–63 (2013)
21. Jiang, Z., Pajic, M., Connolly, A., Dixit, S., Mangharam, R.: A platform for implantable medical device validation: demo abstract. In: WH'10, pp. 208–209. ACM (2010)
22. Méry, D., Singh, N.K.: Formal specification of medical systems by proof-based refinement. ACM Trans. Embed. Comput. Syst. **12**(1), 15:1–15:25 (2013)
23. Méry, D., Singh, N.K.: Ideal mode selection of a cardiac pacing system. In: Duffy, V.G. (ed.) HCII 2013 and DHM 2013, Part I. LNCS, vol. 8025, pp. 258–267. Springer, Heidelberg (2013)
24. Wright, S.: Automatic generation of C from Event-B. In: Workshop on Integration of Model-based Formal Methods and Tools (2009)

Author Index