Dietmar Winkler
Stefan Biffl
Johannes Bergsmann (Eds.)

LNBIP 302

Software Quality

Methods and Tools for Better Software and Systems

10th International Conference, SWQD 2018
Vienna, Austria, January 16–19, 2018
Proceedings

2018 software quality days

EXPERIENCE THE VALUE OF QUALITY

Springer

Dietmar Winkler · Stefan Biffl
Johannes Bergsmann (Eds.)

Software Quality

Methods and Tools for Better Software and Systems

10th International Conference, SWQD 2018
Vienna, Austria, January 16–19, 2018
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 978-3-319-71439-4 ISBN 978-3-319-71440-0 (eBook)
https://doi.org/10.1007/978-3-319-71440-0

Library of Congress Control Number: 2017959622

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Message from the General Chair

The Software Quality Days (SWQD) conference and tools fair started in 2009 and has grown to be the biggest conference 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 software process and quality managers, test managers, software testers, product managers, agile masters, project managers, software architects, software designers, requirements engineers, user interface designers, software developers, IT managers, release managers, development managers, application managers, and similar roles.

The guiding conference topic of SWQD 2018 was "Software Quality 4.0: Methods and Tools for Better Software and Systems," as changed product, process, and service requirements, e.g., distributed engineering projects, mobile applications, involvement of heterogeneous disciplines and stakeholders, frequent changing requirements, extended application areas, and new technologies include new challenges and might require new and adapted methods and tools to support quality assurance activities early.

January 2018 Johannes Bergsmann

Message from the Scientific Program Chair

The 10th Software Quality Days (SWQD) conference and tools fair brought 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 program published in scientific proceedings. For the seventh year we received 16 high-quality submissions from researchers across Europe which were each peer-reviewed by three or more reviewers. Out of these submissions, the editors selected six contributions as full papers, for an acceptance rate of 38%. Furthermore, two short papers, representing promising research directions, were accepted to spark discussions between researchers and practitioners at the conference. This year we invited two scientific keynote speakers for the scientific program, who contributed with two invited papers.

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

This book is structured according to the sessions of the scientific program following the guiding conference topic "Software Quality 4.0: Methods and Tools for better Software and Systems":

- Safety and Security
- Requirements Engineering and Requirements-Based Testing
- Software Requirements and Architecture
- Crowdsourcing in Software Engineering
- Experimentation in Software Engineering
- Smart Environments

January 2018 Stefan Biffl

Organization

SWQD 2018 was organized by Software Quality Lab GmbH and the Vienna University of Technology, Institute of Software Technology and Interactive Systems.

Organizing Committee

General Chair

Johannes Bergsmann	Software Quality Lab GmbH

Scientific Program Chair

Stefan Biffl	Vienna University of Technology

Proceedings Chair

Dietmar Winkler	Vienna University of Technology

Organizing and Publicity Chair

Petra Bergsmann	Software Quality Lab GmbH

Program Committee

SWQD 2018 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 Iceland, Iceland
Ruth Breu	University of Innsbruck, Austria
Maya Daneva	University of Twente, The Netherlands
Oscar Dieste	Universidad Politecnica de Madrid, Spain
Frank Elberzhager	Fraunhofer IESE, Germany
Michael Felderer	University of Innsbruck, Austria
Gordon Fraser	University of Sheffield, UK
Ahmad Nauman Ghazi	Blekinge Institute of Technology, Sweden
Volker Gruhn	Universität Duisburg-Essen, Germany
Roman Haas	Technical University of Munich, Germany
Jens Heidrich	Fraunhofer IESE, Germany
Frank Houdek	Daimler AG, Germany
Slinger Jansen	Utrecht University, The Netherlands

Marcos Kalinowski	Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil
Petri Kettunen	University of Helsinki, Finland
Marco Kuhrmann	Clausthal University of Technology, Germany
Ricardo Machado	CCG-Centro de Computação Gráfica, Portugal
Eda Marchetti	ISTI-CNR, Italy
Emilia Mendes	Blekinge Institute of Technology, Sweden
Daniel Méndez Fernández	Technische Universität München, Germany
Paula Monteiro	Universidade do Minho, Portugal
Juergen Muench	Reutlingen University, Germany
Oscar Pastor Lopez	Universitat Politecnica de Valencia, Spain
Dietmar Pfahl	University of Tartu, Estonia
Rick Rabiser	Johannes Kepler University Linz, Austria
Rudolf Ramler	Software Competence Center Hagenberg, Austria
Andreas Rausch	Technische Universität Clausthal, Germany
Barbara Russo	Free University of Bolzano/Bozen, Italy
Ina Schieferdecker	Fraunhofer FOKUS/TU Berlin, Germany
Miroslaw Staron	University of Gothenburg, Sweden
Rini Van Solingen	Delft University of Technology, The Netherlands
Andreas Vogelsang	Technische Universität Berlin, Germany
Stefan Wagner	University of Stuttgart, Germany
Dietmar Winkler	Vienna University of Technology, Austria

Additional Reviewers

Tiago Amorim
Michael Huber
Clemens Sauerwein

Contents

Safety and Security

Security Challenges in Cyber-Physical Production Systems

Peter Kieseberg[1(✉)] and Edgar Weippl[2]

[1] SBA Research, Vienna, Austria
pkieseberg@sba-research.org
[2] TU Wien, Vienna, Austria
edgar.weippl@tuwien.ac.at

Abstract. Within the last decade, Security became a major focus in the traditional IT-Industry, mainly through the interconnection of systems and especially through the connection to the Internet. This opened up a huge new attack surface, which resulted in major takedowns of legitimate services and new forms of crime and destruction. This led to the development of a multitude of new defense mechanisms and strategies, as well as the establishing of Security procedures on both, organizational and technical level. Production systems have mostly remained in isolation during these past years, with security typically focused on the perimeter. Now, with the introduction of new paradigms like Industry 4.0, this isolation is questioned heavily with Physical Production Systems (PPSs) now connected to an IT-world resulting in cyber-physical systems sharing the attack surface of traditional web based interfaces while featuring completely different goals, parameters like lifetime and safety, as well as construction. In this work, we present an outline on the major security challenges faced by cyber-physical production systems. While many of these challenges harken back to issues also present in traditional web based IT, we will thoroughly analyze the differences. Still, many new attack vectors appeared in the past, either in practical attacks like Stuxnet, or in theoretical work. These attack vectors use specific features or design elements of cyber-physical systems to their advantage and are unparalleled in traditional IT. Furthermore, many mitigation strategies prevalent in traditional IT systems are not applicable in the industrial world, e.g., patching, thus rendering traditional strategies in IT-Security unfeasible. A thorough discussion of the major challenges in CPPS-Security is thus required in order to focus research on the most important targets.

Keywords: Cyber-physical systems · CPS
Cyber-physical production systems · CPPS · Industry 4.0
Advanced manufacturing · Security

1 Introduction

With the continuing digitalization of our economy, even the more traditional branches of the producing industry have become increasingly connected to networks in general, and specifically the Internet. While this brings a lot of new changes with respect to new products and services, it also becomes an increasing challenge for the security of said systems. In the past, several spectacular attacks were launched against industrial environments, most notably the STUXNET [15] malware that aimed at and succeeded in infiltrating a specially sealed environment and carried out an attack that was specifically designed to cause severe damage in an unobtrusive manner. Still, while STUXNET [15] might be the most famous example, typical production systems contain an increasing amount of networking infrastructure, thus becoming cyber-physical systems. This especially means that the systems are not sealed off by a so-called air-gap anymore, but have digital interfaces to the outside world, sometimes even at the component level. Thus, also the attack landscape changed drastically from focusing on getting physical access to a plant towards fully digital attacks carried out through insecure entry points, which are often not even known to the factory operator. In addition, this increasing attack surface is not only a purely financial problem for the operator, but can even have a massive effect on national security in case when infrastructures critical for the nation (e.g., power plants, grid components) are attacked. The recent attacks on the Ukrainian power grid may serve as a perfect example [16]. Thus, in this paper we will discuss the implications of introducing networking equipment into industrial environments from a security perspective. One major focus will be the secure introduction of networks into CPS, while the other major focus lies in protecting the production system engineering process, a topic that has not been in the focus of research until now. While this paper cannot give solutions to most of these problems, we will discuss potential solution strategies that can be the starting point for further research.

The rest of this paper is structured as follows: In Sect. 2 we discuss security issues in industrial environments, including a separation from IoT-systems, which are often mingled. Section 3 deals with the issue of security in the Production System Engineering process (PSE process), concerning both, the introduction of security as a step inside the PSE, as well as securing the PSE itself. The paper is summarized and concluded in Sect. 4.

2 Security Challenges in Industrial Environments

In this section we will discuss major security challenges in cyber-physical systems, especially with respect to their counterparts in traditional web based systems.

2.1 Industrial Environments Versus IoT

When challenging the issue of security in industrial systems, the discussion is often mixed with the challenge of securing products and services based on the "Internet of Things" (IoT), especially when dealing with the Industry 4.0 paradigm. Of course there are certain similarities between IoT and Industry 4.0

installations, namely that they both often rely on the availability of cheap electronics that can be used as sensors, actuators, control mechanisms or other cyber-physical elements. This similarity is often extended to the whole field of introducing security to industrial systems, still, in our opinion there are major differences between cyber-physical production environments and typical IoT-installations that reflect on the security measures that can be enforced.

One major difference lies in the typical life span of either of the two installations. While industrial environments are designed for long life spans typically ranging around 40 to 50 years, IoT installations are seen as more of an end user focused thing with typical consumer life spans of some few years. Of course this is not always the case as e.g., in building automation. Still, this of course does not mean that a production system is never changed during its life span and it must also be taken into account that many parts will be exchanged for spare parts during the lifetime of such an environment, still, many parameters and major parts will still be in place after decades, leading to the problem of heaving to deal with insecure legacy systems, where sometimes even the producer might long have been gone. This issue also leads to the problem of heterogeneity in industrial environments, as decades of development and technical progress might lie between different components inside the same factory, an issue that is currently not typical for IoT-systems.

Another major difference lies in the pricing of the components, where current IoT environments focus on low-cost sensors and modules that can be applied in large quantities, whereas industrial modules are often expensive. This also disallows the exchange of insecure components that cannot be fixed or isolated in industrial environments, as opposed to the typical IoT installation.

Certification is another issue that is very typical for industrial environments, especially regarding the issue of safety, especially when human lives are concerned, which is rather typical for environments like steel mills and similar installations. Most certifications require the owner of the factory to undergo re-certification for every major change, and often even in case of minor changes, applied to the production system. This is not only pertaining to changes on the hardware side, but also in case of software changes, making patching, which is a typical security strategy in the IoT world, very expensive and time-consuming. Thus, while IoT-systems are more similar to software-only systems with respect to patching, industrial environments typically do not have this option. This is additionally reinforced by the issue stated above on legacy systems that often either lack the performance for introducing security measures, or where simply even the knowledge on their internal system workings is lost throughout the decades.

2.2 Challenges Regarding Software Development and Networking

In the modern world of traditional IT and network security, a multitude of different mechanisms and strategies has been devised in order to provide security, as well as to protect vital information. As outlined in Sect. 2.1, several standard security strategies from the traditional software world cannot be applied that easily in cyber-physical production environments.

One major difference to traditional IT systems, and one that is especially critical for cyber-physical systems integrated with the Internet or other accessible network interfaces, is the topic of patching. During the past year, the notion of *Secure Software Development Lifecycles* (SSDLCs) [12] has become a de-facto standard for the development of software, especially in critical environments, but also in many standard applications [18]. One major aspect of these SSDLCs is the notion of patching, i.e., finding and fixing bugs in software that is already shipped to customers and applying the respective changes in the form of software updates. The speed and quality of these fixes is a well-regarded measure for the overall security fitness of the companies behind these products and the removal of products from the SSDLC is typically regarded as discontinuation of said product (e.g., Windows XP). The issues with patching in industrial environments are manifold and ask for completely new strategies:

– Due to safety requirements, quick changes to parts of an industrial environment are typically not allowed, neither on the hardware, nor on the software side. Depending on the regulations of the respective industry, re-certification might become mandatory even in case of very small changes in the software, thus introducing non-negligible costs and, even more problematic in the case of patches, a considerable delay in the application of the patch.
– Due to the considerable life spans of industrial systems, a lot of legacy systems can typically be found in older plants, for which no patches are delivered by the vendors, or even the vendors themselves might not exist anymore.
– Especially in case of legacy systems with their outdated technology, the performance of these systems is often not strong enough to cater for the requirements of modern security measures like strong end-to-end encryption of system messages or digital signatures, as these operations are quite expensive in nature.
– Industrial systems are very complex environments with a multitude of modules interacting with each other. Furthermore, many factories are one of a kind, i.e., the selection of components, their interactions, specifications, but even whole subsystems are unique. Therefore, patching a module inside such a complex system can easily result in side effects that cannot be foreseen when only considering the isolated module. Testing changes for such side effect is practically impossible, changes to the system thus potentially dangerous.
– Another problem is the delivery of the patches: While modern systems often rely on connection to the Internet, this is often not true for industrial plants. Furthermore, many industrial production units do not allow for downtime, which is a major issue with patching, especially when the component in question is not made for being patched at all, i.e., techniques like hot-patching [20] cannot be used.

Thus, one of the major challenges in the area of IT-Security for cyber-physical production systems is the question, how to deal with prominent insecure modules inside the factory. Furthermore, many protocols still in use do not cater for security needs at all. This is often due to the fact that at the time of construction of the plant, security had not been an issue. Introducing security to such a system is often problematic due to time constraints, e.g., in real-time systems, where the overhead introduced by security measures like encrypted data transfer cannot be tolerated. In addition, changes to the communication protocols introduce a lot of additional changes in all the modules that communicate with the module in question, leading to a cascade of additional changes to the system.

Traditionally, this security issue was solved by providing a so-called *air-gap* between the different systems, i.e., the internal systems were not connected to the Internet at all. Still, there are some problems with this strategy:

1. With the introduction of the Industry 4.0 paradigm, an increasing number of modules require connection with each other or even (indirectly) with an outside network.
2. Through the exchange of parts during the regular maintenance processes, new interfaces can be introduced to modules. This is due to the fact that standard chip sets are typically used in order to lower costs. These standard chip sets typically possess a minimum set of capabilities used by many customers, in order to sell high volumes. Thus, a new spare part that was built to the same requirements than the original one might, unbeknownst to the maintenance crew, provide an additional networking interface that can be used by attackers, thus expanding the attack surface.
3. Even in case of providing an air-gap, there have been successful attacks as outlined by the Stuxnet attack, but also other instances [4].

Again, this challenge must be met with additional research on how to deal with systems that were designed to be completely unreachable by external attackers and therefore cannot provide the security measures required for modern interconnected environments. It must be noted that many of these problems are especially prominent in brown-field scenarios, where existing factories with a lot of existing legacy systems are upgraded, while some of these problems might be mitigated in pure green-field scenarios by proactively selecting subsystems that already take care about security issues.

2.3 Attack Detection and Security Models

The detection of attacks is one of the major issues when providing security to systems. In traditional IT systems, Intrusion Detection Systems (IDS) [2] together with anti-virus software are typical measures for attack detection, as well as information gathering on detected attacks. Many forms of intrusion detection have been devised in the past, ranging from pure network profiling to anomaly-based self-learning systems based on modern machine learning algorithms that work by learning “typical behaviour” based on normal usage and then try to detect uncommon traffic that might hint at an attack [6]. While IDSs are a typical measure in modern IT-systems, they can cause problems in industrial environments, especially with the introduction of overhead. This is, again, especially problematic in the case of real-time systems, where even minimal additional overhead is non-negligible and, especially in combination with legacy systems, might lead to damages to the system or even introduce safety issues. The same holds true for scanners scanning for malware on the network or on the machines.

In order to be able to supply a comprehensive analysis of the whole reach of different attacks, typical systems need to be analyzed and generalized into sets of abstract models that can be used for the further analysis. This also includes the modeling of the inherent attributes, especially target controls, side parameters and requirements, in order to provide as generalized models as possible. To the best of our knowledge, there exist no generalized models related to the security of industrial systems to this date.

In order to be able to assess the security of a system based on a model, it is not only required to model the system itself, but also the possible attackers and the attack vectors, as well as the targets of attackers. Especially the latter can be very different from normal IT-systems, as industrial systems are typically far more vulnerable with respect to timing and small delays or small changes in parameters (e.g., temperature control in a steel mill) can cause serious and sometimes irreparable damages. With these differences in targets, also the attacker models will differ strongly compared to the typical attacker models of standard IT-security, which requires further investigation. Furthermore, as industrial systems often possess completely different architectures when compared to normal IT-systems, also the attack vectors require new models.

In addition, the cyber-physical world offers other classes of attacks: Due to the complexity of the cyberphysical world and especially the criticality of operations with respect to timing and other parameters, many new attack scenarios can be devised with major differences to typical goals in IT-Security. For example, typically sensors send information on the state of a physical process to a control unit that regulates the process by steering actuators, thus resulting in complex control loops. Most prominent example is the introduction of manipulated sensor information into feedback loops, as done by the Stuxnet malware [15]. The main issue, why this was such a successful attack, lies in the complexity of the underlying physical process, which is only understood by a small selection of experts. Furthermore, results are not binary in the physical world: The machine does not "work or not work", it might still work in case of a successful attack, but not as optimal, outside of specifications or with higher deterioration. One issue that has been previously identified as important issue is the topic of detection of manipulated sensors. This new class of attacks, which can be considered to be the "attacks of the future" with respect to industrial systems, need to be investigated before they can be modeled in the testing process, since they are relatively new and, to this date, under-researched. Based on these theoretical foundations, models need to be devised that can be incorporated into automated testing processes.

2.4 Securing Data

Often overlooked, even from an expert perspective, the protection of data is a highly critical aspect in industrial systems. This does not only adhere to the classical topic of protecting sensitive personal information in medical industrial environments, but also includes data that is often overlooked: Control information for the industrial process itself, as well as sensor information. Both data streams are vital for the industrial process, not only for the flawless operation, but the optimal setting of parameters is often secret allowing a company to e.g., produce cheaper than its competition. While the control units might be the same for all competing companies on a specific market, the details on the settings thus often constitute a well-protected company secret. With the introduction of networking to these control units, this secret information requires special protection, as it is a valuable target for industrial espionage. Furthermore, an attacker could also gain vital knowledge on the system by monitoring sensor data, as well as even carry out specific attacks by manipulating older sensor data stored. These data assets need to be modeled for any automated testing process. Furthermore, another attack vector that is often overlooked is the testing process itself. When testing systems, they typically need to set all operational parameters in order for realistic testing, which could reveal a lot of industrial knowledge to the tester. Currently, there are no approaches tackling this problem [7].

Today, data in industrial environments is not specially secured, as the main paradigm that was in force when designing the foundations of such systems was that the system (and thus the data) is strictly separated from any outside network, thus making special data protection unnecessary. With the advent of the Industrie 4.0 paradigm, this construction paradigm has changed drastically, still, this has not been reflected much in current designs of data stores [11]. General concepts on how to securely store information for industrial processes, especially considering control and steering parameters which are often neglected by IT-analysts in their risk assessment, thus remains an open research question, including designs for secure data stores that are specifically tailored to the needs and requirements of industrial systems.

Another research aspect related to data protection that requires further academic research is the topic of provable deletion. The topic of provable deletion of data, i.e., forms of deletion that do not allow data restore, has been neglected in Security research for the longest time. Just recently, this topic gained some attention in research (as well as in the industry) due to the General Data Protection Regulation (GDPR) [8] that requires operators to provable delete sensitive private information on request by the owner of said information (typically the data subject). While this feature is already explicitly required by the GDPR, the regulation does not give any further details on this issue, which leaves a huge gap for research: Neither does it state, what provable deletion actually is, nor does it give guidelines on how to achieve a process compliant with the GDPR. Regarding the definition, this is especially a problem for complex systems that store data on many levels and aggregation forms, including backups for disaster recovery. Especially when using more complex mechanisms like databases, which typically need to comply with concepts of data redundancy, crash recovery and so forth, no applicable deletion process exists to this date that works against all known forms of digital forensics, short of physical destruction of the equipment of course [9]. Since testing is often done using real-world data and control information in order to simulate the real-world environment, the test bed amasses sensitive data, which is required to be deleted after the end of the test procedure, or sometimes even after each test run. Thus, it is important to research methods for effective data deletion in complex systems in order to enhance the resilience of the test bed with respect to data leakage.

3 Security Challenges for the PSE Process

One issue that until now has not received the respect and attention it deserves lies within the way plants are designed and constructed, the Production System Engineering (PSE) process. This process lies at the very heart of the development of new production lines (green field), as well as in the processes regarding upgrading and re-designing existing plants (brown field). Especially the latter one becomes increasingly important in an economy that is highly driven by savings and cost-efficiency, as well as by the introduction of new products and services based on the utilization of (sensor) data and other information. In this section we will discuss two major issues, (i) the introduction of security as field of expertise into the PSE process and (ii) the protection of the PSE process itself against manipulation and industrial espionage.

3.1 Current Approaches in Production System Engineering

Planning industrial environments is a very complicated task and requires the incorporation of experts from different fields like mechanical and electrical engineering. In practice, the processes are planned using a waterfall model as described in standard literature, such as the VDI guidelines 2221 [22] and VDI 2206 [21]. Contrary to this, the applied process is often quite different from this strict waterfall model, as changes in subsequent steps need experts involved at an earlier stage to re-iterate their work. In addition to the problem of resulting mismatches between plans and actions, the whole environment has changed considerably in the past years, resulting in increasing parallelization of engineering activities [3] and an increased number of engineering cycles required in order to arrive with a production system model applicable to production system installation [13].

Furthermore, production system engineering tends to solve discipline-specific steps in a predefined (but project-dependent) sequence [10,17], during which the subsequent discipline bases their engineering decisions on the results of the previous steps. Usually, there is no review of previous design decisions. Thereby, it may happen that prior engineering decisions limit the decision options of later development phases. This is especially challenging for the integration of data security considerations as an additional engineering discipline [23], since decisions taken during the production process design, mechanical engineering, and electrical engineering may limit applicable security measures in a way that does not allow to fulfill security requirements at all.

3.2 Introducing Security to PSE

Currently, the development of industrial production systems follows a waterfall model with feedback mechanisms that require a significant manual interaction and lack clear documentation. While this can be a problem for the design process as a whole, it is a fundamental problem for defining a secure environment or putting the required security measures into place, as even very small changes on the functional level in any design step can have a huge impact on the overall system. For example, an engineer might exchange a part for a functionally equivalent one, which just happens to have another chip set integrated that additionally provides a wireless LAN interface while having no impact on any other step. Thus, also last-minute changes on-site during the deployment phase must be tracked and incorporated in the security analysis.

One issue with the current model is the introduction of software at the end of the design chain after all other steps have already been finished, thus not allowing the introduction of hardware-based measures (e.g., de-coupling and unlinking of networks, hardware security appliances) without reverting to a previous step. Also, introducing security at this late step means that many previous design decisions have been made without the notion of security in mind. However, security cannot be introduced in one of the earlier steps alone, as they are lacking the information of the latter design phases, as outlined above. The issue is that every step introduces changes to the overall system that directly reflect on security, therefore, security needs to be considered in each step, including feedback loops to all other steps in order to solve newly arising risks at the most suitable stage. How to introduce such cross-domain solutions into PSE is currently an open research question, next to how to make this process that spans several very different domains, ranging from machine engineering over electronics to IT an agile one.

3.3 Securing the PSE Process

While Sect. 3.2 discussed the challenges of introducing security as a field of expertise into the design process of CPPSs, in this section we will briefly outline the issue of securing the PSE process itself.

The main issue is that, in contrast to the classical example of medical research in hospitals, many experts from several different domains, each with its own set of tools, taxonomies and design obstacles, need to work together, while every change in one of the domains potentially results in changes in the other domains. While this effect is also present in current environments, the problem is gaining another level of complexity when introducing the agile PSE process required for tackling security issues (see Sect. 3.2), as the number of changes going back and forth will increase drastically. The problem is becoming even more complex since, as outlined before, especially in security, seemingly small functional changes can result in the emerging of serious new attack vectors and/or vulnerabilities.

Furthermore, as this issue can boil down to a question of accountability, it must be ensured that the mechanisms provide actual prove as to who is the person responsible. Thus, many open research questions with respect to providing integrity focus on developing mechanisms that provide a provable tracking of changes in the design process. As an additional requirement, many of the experts in such a process have equal rights and, due to the high level of specialization in each domain, there is no central authority available to decide on the different changes. The actual state of the PSE process must be determined in a distributed manner, i.e., all experts have to agree on a specific state as the valid one. On possible solution could lie in the adoption of smart contracts and other blockchain-based approaches, as they provide exactly this functionality in other domains [5]. Furthermore, in order to seal certain parts of the PSE, some form of rights management and access control must be devised that can be applied to a highly dynamic set of assets and users, without requiring a central steering instance. None of the involved parties can assess how and to what extend the knowledge of other domains needs to be protected, i.e., each expert has to have full control over what to share and how to guard the assets. This is an open research issue in the scientific community, but possibly this could be combined with the blockchain based approaches in order to generate smart contracts that provide access control and rights management.

With respect to securing interfaces, the main differences between standard IT systems and the systems prevalent in industrial environments come into play: While standard IT systems change quickly, industrial systems are in use for decades. Also, it is not possible to make significant changes to industrial systems once they are deployed, which also holds true for a lot of industry-specific planning and design software used in PSE. Thus, a secure agile PSE process requires its tools to be shielded from each other, as a weakness in one of them could impact the others, e.g., by extracting knowledge from the PSE process or introducing changes invisible to other parties. While the latter is addressed by new technologies for tracking changes, it is clear that the agile PSE process must itself be secured against weaknesses, opening up a selection of specific research questions that need to be tackled.

3.4 Securing PSE Information

Industrial espionage and the loss of knowledge to competitors is one of the major obstacles in cooperative design projects. This problem is increasing further due to the ongoing specialization w.r.t. the integration of IT technologies into classical systems. Consequently, more and more experts have to get involved in the design process. This development will continue when implementing agile PSE processes, including cross-domain topics like security which extend the amount of interactions while reducing the ability to put functionalities into black boxes (i.e., hiding details). With respect to security, a lot of this hidden functionality must be disclosed in order to analyze and model the attack surface as well as possible attack vectors. Furthermore, certain steps might be outsourced to other partners, either due to cost issues or for political reasons (e.g., mandatory involvement of local subcontractors). If numerous different partners that probably are competitors in other tenders are working on the same project, the protection of know-how is of the utmost importance.

With respect to the protection of knowledge, major research issues which are not satisfactorily solved to date are to be found in automating the measuring of protection requirements w.r.t. derived information, e.g., in the course of aggregations and other transformations. Furthermore, the question of how to abstract planning information in order to provide knowledge protection is unanswered.

One approach to knowledge protection in PSE is the generation of sealed development environments. While such approaches exist in other domains, mainly in the medical sector, the requirements and side parameters regarding such an environment are very different for the PSE process, mainly due to the issue that there is no dedicated data owner facing a selection of data analyses. The agile PSE process requires an environment consisting of many different experts that provide knowledge either directly or indirectly by applying changes and making decisions. This also means that the knowledge to be protected is far more abstract than, e.g., medical data records. Currently it is unclear how to model and quantify this knowledge, therefore we plan to do extensive research on this issue. Furthermore, sealed environments need to provide modular feature sets, i.e., a wide selection of programs of which many are outdated, unsupported, or non-standardized and user knowledge management. Channeling this complexity into a sealed environment while allowing for updates and new features to be introduced is very complicated and a challenging research task.

In addition, there is a certain amount of data that requires strict protection, ranging from actual sensor information akin to data found in traditional medical environments to meta-information and algorithms to specific configuration information that possesses great value for the owner. In order to protect these types of information, research in the area of data leak protection is required. In contrast to other data-preserving technologies, data leak protection aims at provably finding data leaks, i.e., users having access to data and distributing it illegally. Two basic concepts have been proven to be especially promising, which are (i) the use of obfuscation/diversification and (ii) the concept of watermarking.

The basic idea behind obfuscation lies in making the information unintelligible for any attacker, thus removing its value. While typically used to protect code, i.e., by generating a version of the program that cannot be reversed easily [19], there are also strategies to obfuscate data [24]. The main issue with obfuscation, however, is that compared to cryptography the strength of the obfuscation is typically not measurable [19]. Furthermore, these techniques target normal IT-environments and do not take the requirements of industrial systems, e.g., performance-wise, into account. In addition, they target non-collaborative environments where one side is seen as passive user, which is completely different from partners in a cooperative development environment. Still, obfuscation can play a vital part in the protection of algorithms, as well as diversifying software environments, i.e., making the execution paths of a software installation depending on external parameters like passwords, user names or even control information in order to reduce the danger of class breaks. While these techniques exist for traditional IT environments, there is still a lot of research to be conducted for applying them in industrial environments.

Data leak detection through watermarking follows a completely different concept. While defensive mechanisms like obfuscation and anonymization strife to remove the value of the leaked information to the attacker (proactive measures), watermarking is a reactive measure that aims at identifying exfiltrated data in order to support legal measures. One of the main reasons is that even anonymized data might still hold significant value and it might not be possible to reduce the quality of information distributed to cooperating experts without destroying the cooperation. Typical approaches in the literature aim at protecting large amounts of structured data [25], e.g., records in a medical database. Many of these approaches work by inserting additional data, e.g., by inserting new or changing existing tuples [1] or by using the intrinsic features of the anonymization process [14]. Still, these approaches do not work for the full range of information that needs protection in industrial environments, especially not for control or configuration information, which is often only of very small size and very sensitive to even minor changes or loss of granularity as introduced by the types of algorithms outlined above.

4 Conclusion

In this work we discussed the major issues of providing security in cyber-physical production systems. The focus of our analysis was directed into two major directions, (i) the problem of introducing security into CPPS and the differences to securing traditional IT-systems and (ii) introducing security into the production system engineering process (PSE). Especially for the latter, we not only discussed the issue of designing secure systems within an agile process, but we also discussed on how to secure the process itself with respect to manipulation and industrial espionage. Both topics have been thoroughly neglected by academics until now, but need to be considered as important issues when discussing the hardening of critical systems.

In conclusion, the topic of providing in-depth security at a professional level, comparable to the one achievable in traditional software environments, in cyber-physical production systems still requires large efforts on the academic, but also the industrial side. While in this work we focused on academic research questions, it must also be kept in mind that the work to adapt the results of such research into real-life applications requires a lot of effort from the side of the industry. With dwindling profits in both, the producing industries, as well as in the production systems engineering industry, this will remain a problem. Still, as many of these cyber-physical systems also represent critical national infrastructure, solutions for securing these systems need to be devised and implemented soon.

References

1. Agrawal, R., Kiernan, J.: Watermarking relational databases. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 155–166. VLDB Endowment (2002)
2. Barry, B.I.A., Chan, H.A.: Intrusion detection systems. In: Stavroulakis, P., Stamp, M. (eds.) Handbook of Information and Communication Security, pp. 193–205. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-04117-4_10
3. Barth, M., Biffl, S., Drath, R., Fay, A., Winkler, D.: Bewertung der offenheit von engineering-tools. Open Autom. **4**(13), 12–15 (2013)
4. Byres, E.: The air gap: SCADA's enduring security myth. Commun. ACM **56**(8), 29–31 (2013)
5. Crosby, M., Pattanayak, P., Verma, S., Kalyanaraman, V.: Blockchain technology: beyond bitcoin. Appl. Innov. **2**, 6–10 (2016)
6. Depren, O., Topallar, M., Anarim, E., Ciliz, M.K.: An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. Expert Syst. Appl. **29**(4), 713–722 (2005)
7. Diemer, J.: Sichere industrie-4.0-plattformen auf basis von community-clouds. In: Vogel-Heuser, B., Bauernhansl, T., ten Hompel, M. (eds.) Handbuch Industrie 4.0 Bd. 1. VDI Springer Reference, pp. 177–204. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-45279-0_39
8. Interinstitutional File: Proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (general data protection regulation) (2012)
9. Villaronga, E.F., Kieseberg, P., Li, T.: Humans forget, machines remember: artificial intelligence and the right to be forgotten. Comput. Secur. Law Rev. **8** (2017)
10. Hell, K., Lüder, A.: Wiederverwendung im engineering. ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb **111**(6), 337–341 (2016)
11. Kagermann, H.: Recommendations for Implementing the Strategic Initiative Industrie 4.0. Forschungsunion, Essen (2013)
12. Howard, M., Lipner, S.: The Security Development Lifecycle, vol. 8. Microsoft Press, Redmond (2006)
13. Hundt, L., Lüder, A.: Development of a method for the implementation of interoperable tool chains applying mechatronical thinking—use case engineering of logic control. In: 2012 IEEE 17th Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8. IEEE (2012)
14. Kieseberg, P., Schrittwieser, S., Mulazzani, M., Echizen, I., Weippl, E.: An algorithm for collusion-resistant anonymization and fingerprinting of sensitive microdata. Electron. Mark. **24**(2), 113–124 (2014)
15. Langner, R.: Stuxnet: dissecting a cyberwarfare weapon. IEEE Secur. Priv. **9**(3), 49–51 (2011)
16. Liang, G., Weller, S.R., Zhao, J., Luo, F., Dong, Z.Y.: The 2015 ukraine blackout: implications for false data injection attacks. IEEE Trans. Power Syst. **32**(4), 3317–3318 (2017)
17. Lindemann, U.: Methodische Entwicklung Technischer Produkte: Methoden Flexibel und Situationsgerecht Anwenden. Springer, Heidelberg (2006). https://doi.org/10.1007/978-3-642-01423-9
18. McGraw, G.: Software security. IEEE Secur. Priv. **2**(2), 80–83 (2004)

19. Nagra, J., Collberg, C.: Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection. Pearson Education, London (2009)
20. Ramaswamy, A., Bratus, S., Smith, S.W., Locasto, M.E.: Katana: a hot patching framework for elf executables. In: International Conference on Availability, Reliability, and Security, ARES 2010, pp. 507–512. IEEE (2010)
21. Richtlinie, V.D.I.: 2206: Entwicklungsmethodik für mechatronische Systeme. VDI-Verlag, Düsseldorf (2004)
22. Richtlinie, V.D.I.: 2221 (1993): Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte. VDI-Verlag, Düsseldorf (2007)
23. Riel, A., Kreiner, C., Macher, G., Messnarz, R.: Integrated design for tackling safety and security challenges of smart products and digital manufacturing. CIRP Ann.-Manuf. Technol. **66**, 177–180 (2017)
24. Schrittwieser, S., Katzenbeisser, S., Kinder, J., Merzdovnik, G., Weippl, E.: Protecting software through obfuscation: can it keep pace with progress in code analysis? ACM Comput. Surv. (CSUR) **49**(1), 4 (2016)
25. Sion, R., Atallah, M., Prabhakar, S.: Watermarking relational databases (2002)

Monitoring of Access Control Policy for Refinement and Improvements

Antonello Calabró, Francesca Lonetti, and Eda Marchetti(✉)

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo",
Consiglio Nazionale delle Ricerche (CNR), via G. Moruzzi 1, 56124 Pisa, Italy
{antonello.calabro,francesca.lonetti,eda.marchetti}@isti.cnr.it

Abstract. Access Control is among the most important security mechanisms to put in place in order to secure applications, and XACML is the de facto standard for defining access control policies. As systems and resource utilization evolve, access control policies become increasingly difficult to manage and update according to contextual behaviour. This paper proposes a policy monitoring infrastructure able to identify policy abnormal behaviour and prevent misuse in granting/denying further accesses. This proposal relies on coverage adequacy criteria as well as KPIs definition for assessing the most common usage behaviors and provide feedback for refinement and maintenance of the current access control policy. It integrates a flexible and adaptable event based monitoring facility for run time validation of policy execution. A first validation on an example shows the effectiveness of the proposed approach.

Keywords: Monitoring · Coverage criteria · KPI · Access control policy

1 Introduction

In the Factory of the Future as well as Industry 4.0 more often the adopted context-aware applications rely on a widespread use of wireless sensors and actuators to monitor the process evolution, guide the users interaction and automate the business process at large. The aggregation of high volumes of volatile data and sensitive user information rises a multitude of security and privacy challenges. Typically, large organizations have to rule millions of accesses across thousands of IT resources and sensors. This cause an effective risk of providing users with an excessive or not appropriate access rights over time. In such a situation the proper implementation and enforcement of access control mechanisms becomes imperative. They are able to rule the highly connected and pervasive software intensive systems and prevent unauthorized, erroneous or even malicious usage of critical resources.

Among security mechanisms, a critical role is played by access control systems. These aim at ensuring that only the intended subjects can access the protected data and the subjects get only the permission levels required to accomplish their tasks and no more. To this purpose, access control systems involve three related activities, namely: *identification* of the user or service requesting an access; *authentication* of the declared identity, usually by means of credentials; and finally *authorization* of authenticated users to perform a set of allowed operations.

Authorization mechanisms are typically based on *access control policies* that rule: the level of confidentiality of data; the procedures for managing data and resources; the classification of resources into category sets yielding different security requirements. Access control policies must be specified and verified with great accuracy: as any error or overlook could result either in forbidding due access rights, or worse in authorizing accesses that should be denied, thus jeopardizing the security of the protected data.

XACML (eXtensible Access Control Markup Language) [1] is a widely used standard language for specifying access control policies. It is an XML-based language conceived with interoperability, extensibility, and distribution in mind, thus enabling the specification of very complex rules. However, such advantages are paid in terms of complexity and verbosity. Writing XACML policies is hard and may be deceiving, as inconsistencies could arise between the security requirements the policy authors intend to specify, and those that the policies actually state. This can easily happen for instance when either some modifications are introduced in a complex policy or when the policy is obtained from the integration of more policies coming from different organizations.

Especially in large scale organizations the common practice, to partially solve this problem, is to define and implement just the basic policies, usually extracted by the internal regulations and requirements that remain unchanged in time. The side effect of such an attitude is that policies could become outdated over time, leading either inconsistencies with the evolved behavioral and technological organization environment, or security vulnerabilities.

So far most, of the research activities have been mainly focused on policy testing [2–5] and only few proposals target the on-line validation and improvement of the policies specifications [6].

The idea of this paper is providing to the policy authors, or more in general to any stakeholder using the policy (generically referred in the following as the policy validators), an integrated dynamic framework to validate and monitor the actual resources access and users behavior. This in order to update, correct or improve the policy specification in granting/denying the accesses.

Through the proposed framework the policy validators may have: (i) a better knowledge of the actual policy usage, (ii) define and on-line calculate possible key performance indicators (KPIs) useful to adjust the policies and (iii) detect possible policy vulnerabilities or outdated rules.

The proposal relies on: (i) the derivation of the relevant coverage information from the policy specification; (ii) the collection of events (requests/responses) during the policy execution by means of a monitoring facility, (iii) and the analysis of them so to collect important policy accesses information and assess defined KPIs.

Indeed monitoring the different requests and responses lets the collection of source data such as: the identification of most accessed resources; the tracking of type of users requiring a resource access; the time and the decision of whether the request is granted or denied and so on. These can be very useful data for modeling the most common user behaviors and detecting possible policy flaws or improvements. Moreover key performance indicators, relative to the criticality of some specific actions or data, can be specified and on-line computed for a better resources access control. For instance possible KPIs could be: the frequency of the resource access by a specific users; the identification of users that most frequently require a specific action; the date and time when the access permission of a specific resource is more frequently activated and so on.

Deviation of the observed data from the boundaries values established for each KPI could indicate abnormal behavior and could required a deeper analysis to identify possibly access control policy modifications or improvements.

The contribution of this paper can be summarized into:

– the integration of a monitoring framework into an access control system architecture;
– the definition of the architecture of the Policy Monitoring Infrastructure enabling: the definition of specific KPIs; the policy parsing so to extract coverage information; the monitoring of requests and response execution so to compute coverage data; the analysis of data collected to assessing their compliance with the access control rules and specified KPI;
– an instantiation of the proposed architecture on the XACML access control language.

An simulation example showing the usage of the proposed monitoring framework is also provided considering a prototyped version of a booking system of the Multimedia Laboratory of Pisa university, used for improving foreign languages or preparing lecture material.

Preliminary results of such example demonstrated the feasibility of the proposal and highlighted some of the potentialities of the Policy Monitor Infrastructure. Indeed without such a feature most of the weaknesses evidenced in the experiment and suggestions for access control policy improvements were not easily identifiable.

The remainder of this paper is structured as follows: Sect. 2 introduces the basic concepts of access control systems and coverage testing; Sect. 3 presents the architecture of the Policy Monitoring Infrastructure; Sect. 4 reports of the usage of the Policy Monitoring Infrastructure on example; Related works are presented in Sect. 5 and finally, Sect. 6 concludes the paper also hinting at discussion and future work.

2 Background

In the following section some basic concepts about XACML access control systems and coverage testing are provided.

2.1 XACML and Access Control Systems

Access control is one of the most adopted security mechanisms for the protection of resources and data against unauthorized, malicious or improper usage or modification. It is based on the implementation of access control policies expressed by a specific standard such for instance the widely adopted eXtensible Access Control Markup Language (XACML) [1].

XACML is a platform-independent XML-based language for the specification of access control policies. Briefly, an XACML policy has a tree structure whose main elements are: PolicySet, Policy, Rule, Target and Condition. The PolicySet includes one or more policies. A Policy contains a Target and one or more rules. The Target specifies a set of constraints on attributes of a given request. The Rule specifies a Target and a Condition containing one or more boolean functions. If the Condition is evaluated to true, then the Rule's Effect (a value of Permit or Deny) is returned, otherwise a NotApplicable decision is formulated (Indeterminate is returned in case of errors). The PolicyCombiningAlgorithm and the RuleCombiningAlgorithm define how to combine the results from multiple policies and rules respectively in order to derive a single access result. The structure of an access control policy and an access control request is sketched in Fig. 1. While an example is provided in Listing 1.

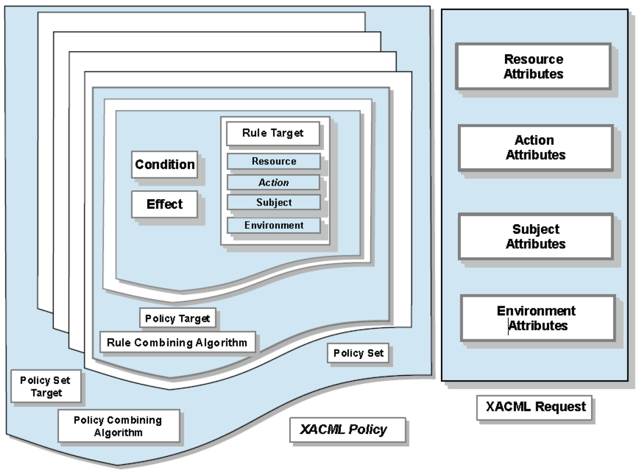


Fig. 1. Anatomy of an XACML policy and an XACML request

The main components of an XACML based access control system are shown in Fig. 2. In particular the Policy Administration Point (PAP) is the system entity in charge of managing the policies; the Policy Enforcement Point (PEP), usually embedded into an application system, receives the access request in its native format, constructs an XACML request and sends it to the Policy Decision Point (PDP); the Policy Information Point (PIP) provides the PDP with the values of subject, resource, action and environment attributes; the PDP evaluates the policy against the request and returns the response, including the authorization decision to the PEP.

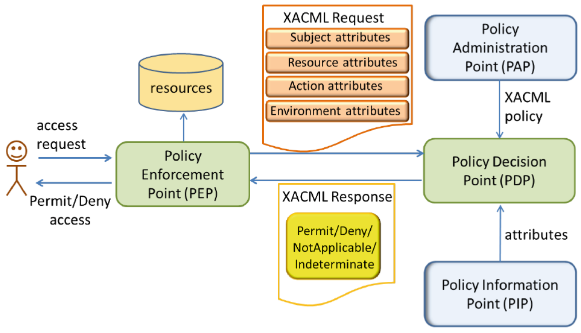


Fig. 2. Access control system architecture

2.2 Adequacy Criteria and Coverage

The notion of adequacy criteria has been extensively investigated in software engineering literature, an in particular for software testing where it is generally used to assess the effectiveness of a test suite [7,8]. In test coverage criteria, a set of requirements that a test suite must fulfill is established and it is mapped onto a set of entities that must be covered when the test cases are executed, as for instance all statements or all branches of a program control-flow. The coverage criterion is satisfied if all the entities are covered; otherwise, the percentage of covered entities represents a quality measure of the test suite.

The intuitive motivation behind measuring test coverage is that if some entity has never been tested, it might contain undetected faults. Obviously, the converse reasoning does not apply: if we had covered all entities and detected no failures, this does not necessarily imply that the program is correct. In a similar way, we propose here to assess the adequacy of the access control policy execution by identifying what are the relevant entities to be covered and by assessing if all of them, or otherwise what percentage, have been executed.

As for test adequacy, the motivation behind assessing access control policy execution adequacy is that if some entities are not covered, we cannot exclude that these might hide some problems, policy incompleteness, entity misuses or security flaw. Similarly to the testing session, i.e. the period along which the test adequacy is measured, the observation window is the observation period associated to the access control policy execution coverage measure.

Intuitively, a sliding observation window over a time measurement unit can be established, which could be either continuous (e.g. the entities covered in the last week) or discrete (e.g., the most recent 15 entities). The proposed access control policy adequacy criterion extends the general monitoring adequacy criterion presented in [9,10], by defining and implementing an instantiation of this adequacy criterion for the XACML access control policy execution. In particular we consider the following definition:

Definition 1. *Denote $r_i \in R$ the i-th entity to be covered, and by $\delta_i \in \Delta$ the length of its associated observation window. The access control policy execution adequacy criterion C dynamically measures the coverage on R for a given entity i at each time unit t as follows:*

$$C[R, \Delta](t) = \frac{\sum_{i=1}^{|R|} \lambda_i(t)}{|R|}$$

where for $r_i \in R$ and $\delta_i \in \Delta$

$$\lambda_i(t) = \begin{cases} 1 & \text{if } r_i \text{ is covered at least once in } [t - \delta_i, t] \\ 0 & \text{otherwise} \end{cases}.$$

According to this definition the length of δ_i could be different for each r_i, or could be the same for all entities. In summary the definition of access control policy execution adequacy introduces the following concepts:

– an "adequate access control policy execution" is a policy execution on which a set of entities r_i are covered in a window δ_i;
– a Policy Monitoring, i.e. an infrastructure that, at every instant t, can provide a coverage measure as in Definition 1. If this is less than 1, it can provide a list of those entities that have not been covered;
– an entity that is not covered is an entity of the access control policy that has not been executed for some time.

Inside a access control policy execution what is an entity to be covered can be provided at different levels and with different targets [2,11]. In this paper we consider the *XACML smart coverage* approach presented in [11] which focuses on the XACML policy rules coverage. Briefly, the criterion computes the *Rule Target Set*, i.e. the union of the target of the rule, and all enclosing policy and policy sets targets. The main idea is that in order to match the rule target, the requests must first match the enclosing policy and policy sets targets. If the rule target has several subjects, resources, actions, and environments and the enclosing policy and policy set targets are empty, to cover the rule target the request should have specifical structure. It should include a subject contained in target subjects set, a resource contained in the target resources set, an action contained in the target actions set, an environment contained in the target environments set. Finally, if the *Rule Target Set* of a rule is empty and its condition is evaluated to True or False, all requests are covering this rule. We refer to [11] for further details.

In this paper, according to this criterion, we adopted the following definitions:

Definition 2 (Rule Entity). *Given a XACML access control policy, a Rule R_i and its* Rule Target Set*, a rule entity RE_i is the couple (*Rule Target Set, Rule Verdict*) where the* Rule Verdict *is the verdict associated to the Rule R_i when its condition is evaluated to True.*

Definition 3 (Rule Coverage Domain). *Considering a XACML access control policy, the Rule Coverage Domain is the set of all the Rule Entities of the policy.*

Definition 4 (Percentage of Rule Coverage). *With reference to Definition 1, the percentage of rule coverage at time t is given by 100*C, where R is the Rule coverage domain.*

Consequently, at a given instant a XACML access control policy execution is adequated with respect to the rule coverage criterion if the percentage of Rule Entities covered is 100% (or greater than an established threshold level).

3 Policy Monitoring Infrastructure

With reference to Fig. 3, we present in this section the components of the Policy Monitoring Infrastructure. In particular:

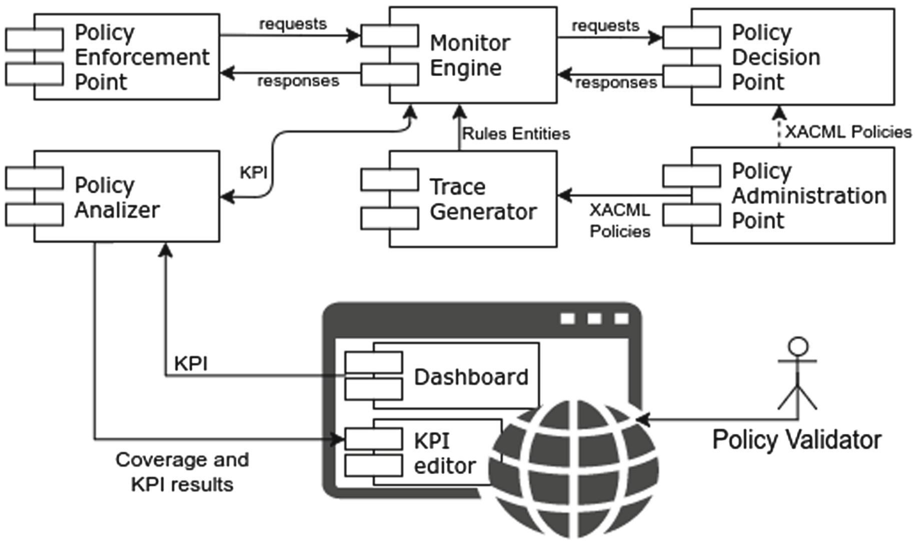


Fig. 3. Policy monitoring infrastructure

– *Policy Enforcement Point (PEP).* It is usually embedded into an application system, receives the access request in its native format, constructs an XACML request and sends it to the Policy Decision Point (PDP) through the *Monitor Engine*;
– *Policy Administration Point (PAP).* It is the system entity in charge of storing and managing the XACML policies. It sends the policy both to the Policy Decision Point (PDP) for its evaluation and to the *Trace Generator* for the traces extraction.

- *Policy Decision Point (PDP).* It evaluates the policy against the request and returns the response, including the authorization decision, to the *Monitoring Engine*. It communicates with the *Monitoring Engine* though a dedicated interface such as a REST one.
- *Trace generator.* It is in charge of implementing the proposed access control policy adequacy criterion. It takes in input the policy from the *Policy Administration Point* and, according to the coverage criterion, derives all the possible Rule Entities. Usually, the Rule Entities extraction is realized by an optimized unfolding algorithm that exploits the policy language structure. Intuitively, the main goal is to derive an acyclic graph, defining a partial order on policy elements. Several proposals are available such as [2,11] for XACML policy specification. Once extracted, the Rule Entities are provided to the *Monitor Engine*.
- *Monitor Engine.* It is in charge of collecting data of interest during the runtime policy execution. There can be different solutions for monitoring activity. In this paper we rely on Glimpse [12] infrastructure which has the peculiarity of decoupling the events specification from their collection and processing. The main components of the *Monitoring Engine* are: (i) the *Complex Events Processor (CEP)* which analyzes the events and correlates them to infer more complex events; (ii) the *Rules Generator* that generates the rules using the rule templates starting from the derived Rule Entities to be monitored. A generic rule consists of two main parts: the events to be matched and the constraints to be verified, and the events/actions to be notified after the rule evaluation; We refer to [12] for a more detailed description of the Glimpse architecture.
- *Policy Analyzer.* It is in charge of the final derivation of the KPIs values according to the covered and non covered Rule Entities. Moreover, according to the coverage criterion it is able to identify the Rule Entities of the policy that could generate flaws or security violations, providing hints to the Policy Validator for policy improvement or refinement.
- *GUI.* It allows to define the KPIs to be monitored (*KPI editor*) and to visualize to the Policy Validators the collected coverage data and KPI values (*Dashboard*) so to let them to analyze and possibly refine the policy. KPI Editor exploits the Domain Specific Language feature of the Drools technologies. In particuar it extracts from the policy specificion some set of values (for instance enviroment condition parameters, subjects, resources, actions an other target contraints) and provides a set of predefined fuctions and logical operators useful for the definition of the KPIs. The purpose is to let the user able to compose rules using a familiar set of logical expressions without requiring knowledge of the implementation language of the Monitor Engine.

4 Example

In this section we present an example of instantiation and use of the Policy Monitoring Infrastructure depicted in Sect. 3. In particular, in the following subsections we briefly introduce the example considered (Sect. 4.1) and discuss the results collected (Sect. 4.2).

4.1 Example Description

The example considered in this paper is a prototyped version of a booking system of the Multimedia Laboratory of Pisa university, used for improving foreign languages or preparing lecture material. Since the number of stations of this laboratory is limited to 25, the access to different types of users (Master Students, PhD Students, Professors) during the working day is regulated by an access control policy which automatically distributes the booking requests over three time slots. Listing 1 presents the simplified version of the XACML access control policy used for booking a station in the Multimedia Laboratory. Specifically, the policy includes a policy target (lines 8–29) allowing the access only for booking the *MultimediaLab* resource. A first rule (ruleA) (lines 30–64) specifies that Master Students can book the lab starting from 9 am for 4 h. A second rule (ruleB) (lines 65–99) specifies that PHD Students can book the lab starting from 14 pm for 4 h, while a third rule (ruleC) (lines 100–134) specifies that Professors can book the lab only after 18 pm. Finally, the default rule (line 135) denies the access in the other cases. Users of the Multimedia Laboratory are not aware of this internal access control policy and through associated PEP they can ask for booking in all the working day time slots.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
3   xmlns:xacmlcontext="urn:oasis:names:tc:xacml:2.0:context:schema:os"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os␣http://
        docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
6   PolicyId="MultimediaLabPolicy"
7   RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
        algorithm:first-applicable">
8    <Target>
9     <Resources>
10     <Resource>
11      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
            equal">
12       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            MultimediaLab</AttributeValue>
13       <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0
            :resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#
            string"/>
14      </ResourceMatch>
15     </Resource>
16    </Resources>
17    <Actions>
18     <Action>
19      <ActionMatch
20       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
21       <AttributeValue
22        DataType="http://www.w3.org/2001/XMLSchema#string">booking</
            AttributeValue>
23       <ActionAttributeDesignator
24        AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
25        DataType="http://www.w3.org/2001/XMLSchema#string"/>
26      </ActionMatch>
27     </Action>
28    </Actions>
```

```
  </Target>
  <Rule RuleId="rule1" Effect="Permit">
   <Description> A subject who is a Master Student can book the MultimediaLab
        in the morning.</Description>
   <Target>
    <Subjects>
     <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
           equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
        >MasterStudent</AttributeValue>
       <SubjectAttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </SubjectMatch>
     </Subject>
    </Subjects>
    <Environments>
     <Environment>
      <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:time-
           equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">09
            :00:00</AttributeValue>
       <EnvironmentAttributeDesignator
        AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
        DataType="http://www.w3.org/2001/XMLSchema#time"/>
      </EnvironmentMatch>
     </Environment>
     <Environment>
      <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0
           :function:dayTimeDuration-equal">
       <AttributeValue
        DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
             dayTimeDuration">PT4H</AttributeValue>
       <EnvironmentAttributeDesignator
        AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
        DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
             dayTimeDuration"/>
      </EnvironmentMatch>
     </Environment>
    </Environments>
   </Target>
  </Rule>
  <Rule RuleId="ruleB" Effect="Permit">
  <Description> A subject who is a PhD Student can book the MultimediaLab in
      the afternoon.</Description>
  <Target>
   <Subjects>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
          equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
       >PHDStudent</AttributeValue>
      <SubjectAttributeDesignator
       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"/>
     </SubjectMatch>
    </Subject>
   </Subjects>
   <Environments>
    <Environment>
     <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:time-
          equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">14
           :00:00</AttributeValue>
      <EnvironmentAttributeDesignator
       AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
       DataType="http://www.w3.org/2001/XMLSchema#time"/>
     </EnvironmentMatch>
    </Environment>
    <Environment>
     <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0
          :function:dayTimeDuration-equal">
      <AttributeValue
       DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
            dayTimeDuration">PT4H</AttributeValue>
      <EnvironmentAttributeDesignator
       AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
       DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
            dayTimeDuration"/>
      </EnvironmentMatch>
     </Environment>
    </Environments>
   </Target>
  </Rule>
  <Rule RuleId="ruleC" Effect="Permit">
  <Description> A subject who is a Professor can book the MultimediaLab in
      the evening.</Description>
  <Target>
   <Subjects>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
          equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"
       >Professor</AttributeValue>
      <SubjectAttributeDesignator
       AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"/>
     </SubjectMatch>
    </Subject>
   </Subjects>
   <Environments>
    <Environment>
     <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:time-
          equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">18
           :00:00</AttributeValue>
      <EnvironmentAttributeDesignator
       AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
       DataType="http://www.w3.org/2001/XMLSchema#time"/>
     </EnvironmentMatch>
    </Environment>
    <Environment>
     <EnvironmentMatch MatchId="urn:oasis:names:tc:xacml:1.0
          :function:dayTimeDuration-equal">
      <AttributeValue
       DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
            dayTimeDuration">PT4H</AttributeValue>
      <EnvironmentAttributeDesignator
       AttributeId= "urn:oasis:names:tc:xacml:1.0:environment:current-time"
       DataType="http://www.w3.org/TR/2002/WD-xquery-operators-20020816#
            dayTimeDuration"/>
     </EnvironmentMatch>
    </Environment>
   </Environments>
  </Target>
  </Rule>
  <Rule RuleId="default" Effect="Deny"/>
</Policy>
```

Listing 1. MultimediaLab access policy

In this section we show the use of the Policy Monitoring Infrastructure schematize in Fig. 3, for analyzing the load of the booking Multimedia Laboratory system during a working day. For this we set up an experiment in which we simulated the behavior of different types of users (Master Students, PhD Students, Professors) in booking the Multimedia Laboratory according to their preferred time slots. In particular considering the Definition 2 of Sect. 2 and the policy of Listing 1, through the *Trace Generator* component, the Policy Monitoring Infrastructure automatically derives the set of Rule Entities as reported in Table 1.

Successively, thought the KPI Editor of the Policy Monitoring Infrastructure GUI, we defined the following KPIs:

- KPI1 = daily percentage the overall accesses of Master Students, PhD Students and Professors greater than 60%
- KPI1.1 = daily percentage of accesses of Master Students greater than 36%
- KPI1.2 = daily percentage of accesses of PhD Students greater than 6%
- KPI1.3 = daily percentage of accesses of Professors greater than 18%
- KPI2 = daily number of allowed requests in each time slot less or equal than 25.

From an implementation point of view the above defined KPIs have been translated into the following rules:

- KPI1 = daily percentage of coverage of RE_1 RE_2 and RE_3 greater than 60%
- KPI1.1 = daily percentage of coverage of RE_1 greater than 36%
- KPI1.2 = daily percentage of coverage of RE_2 greater than 6%
- KPI1.3 = daily percentage of coverage of RE_3 greater than 18%
- KPI2 = daily number of allowed requests in each time slot less or equal than 25.

Table 1. *Rule Coverage Domain* of Listing 1

$RE_1 = \{(\emptyset, \{MultimediaLab\}, \{booking\}, \emptyset),(\{MasterStudent\}, \emptyset, \emptyset, 9-13), Permit\}$
$RE_2 = \{(\emptyset, \{MultimediaLab\}, \{booking\}, \emptyset),(\{PhDStudent\}, \emptyset, \emptyset, 13-17), Permit\}$
$RE_3 = \{(\emptyset, \{MultimediaLab\}, \{booking\}, \emptyset),(\{Professor\}, \emptyset, \emptyset, 17-21), Permit\}$
$RE_4 = \{(\emptyset, \{MultimediaLab\}, \{booking\}, \emptyset),(\emptyset, \emptyset, \emptyset, \emptyset), Deny\}$

Finally to set up the simulation environment we established the frequency of a booking request of the three type of users. For this we took the data relative to the requests to the Multimedia Laboratory booking system of the last five years from people belonging to the Software Engineering course of University of Pisa. Then we derived the following percentages: 60%, 30%, 10% for Master Students, PhD Students, Professors respectively. Moreover, we made interviews to a sample of Professors (10), Master Students (60) and PhD Students (30) asking them their preferred time slots. According to their preferences, we derived the average values of frequencies of preferred time slots as shown in Table 2.

The collected information has been used to derive the sample of booking requests useful for the simulation experiment as reported in the last column of Table 2. Specifically for each of the time slot and according to the computed percentages distribution, we manually derived an overall sample of 1000 XACML requests distributed as in the Table. Each of these requests includes a subject (values chosen from Master Student, PhD Student, Professor), a resource (MultimediaLab), an action (booking) and a time slot (values chosen from 17–21, 13–17, 9–13).

Knowing that the booking system of the Multimedia Laboratory receives on average 50 booking requests per day, through the Policy Monitoring Infrastructure GUI, we set the observation windows of our simulation experiment to a day long (see Definition 1 in Sect. 2). Then we randomly selected, from the generated 1000 XACML requests, sets of 50 requests. We repeated the selection considering a period of 5 weeks, i.e. 25 observation windows, for a overall number of 1250 executed requests.

Bypassing in this simulation experiment the PEP Component, the Monitor Engine Component sent the requests of one by one to the PDP of the Multimedia Laboratory booking system and collected the coverage data.

Table 2. Request frequency

Type of users	Percentage frequency	Time slot	Preferred percentage	# Generated requests
Master students	60%	17–21	60%	360
		13–17	35%	210
		9–13	5%	30
Professors	10%	17–21	5%	5
		13–17	80%	80
		9–13	15%	15
PhD students	30%	17–21	20%	60
		13–17	70%	210
		9–13	10%	30

4.2 Results

In this section we report the results obtained from the simulation experiment. As shown in Table 3, we run on the PDP 50 randomly selected requests per day and we collected, through the Monitor Engine, the data useful for evaluating the KPIs and deriving interesting information on the Multimedia Laboratory booking profiles.

Specifically in the first column there is the incremental number of observation windows considered in the experiment (25 days). From the second to the forth column (from the fifth to the seventh and from the eighth to the tenth) there are the data relative to the Master Students (Professors and PhD Students respectively) booking requests for defined time slots (17–21, 13–17, 9–13). These data have been collected by the Monitor Engine component by comparing and matching the Rule Entities values with the XACML requests and the corresponding PDP responses. These values have been then refined and analyzed the Policy Analyzer Component to derive the KPIs values as reported in Table 3.

In particular in the last four columns there are the daily values computed for the KPI1, KPI1.1, KPI1.2 and KPI1.3 respectively. Finally in the columns highlighted in gray (fourth, fifth, ninth column with bottom label KPI2) there are the comupted KPI2 values corresponding to the daily number of allowed requests in each time slot. Average percentage value for each column is provided in the last row. Results of Table 3 are provided to the Policy Validator through the Dashboard component of the Policy Monitoring Infrastructure GUI.

From the analysis of the data collected during the monitoring activity the Policy Validator could derive differen information such for instance:

- the KPI1 is never satisfied because in each observation windows the coverage percentage is always less than 41% (maximum reached value) with an average value of 29.07% quite far from the established boundary of 60%. This means that the current implemented access control policy does not mach the real booking behavior of the three different users: few booking requests accepted and an extremely high value rejected. Indeed (by difference) the average percentage of coverage of Rule Entity RE_4 is 70.93%. This situation evidences a pressing need to improve the access control policy to better satisfy the real users behavior.
- From a detailed analysis of the data collected of KPI1.1, KPI1.2 and KPI1.3 emerges that the failure of KPI1 is mainly due to the behavior of Master Students and Professors. Indeed, while KPI1.3 is most of the times satisfied (a part from Day 9 and Day 19 where the percentage of coverage are 16% and 13.33% respectively) demonstrating a good ruling of booking access for the PhD Students, KPI1.1, KPI1.2 are never verified. Percentage of accepted booking requests for Master Students rarely reaches values greater than 6% (only Days 10 and 21) and is on average 2.55%. These values are very far from the established 36% of the KPI1.1. Almost the same situation can be experienced for Professors, where the greater value for KPI1.2 is 4.17% with an average of 0.82%. Therefore, improvements on the access control policy for the booking slot times for Master Students and Professors are necessary.
- Considering the daily number of allowed requests in each time slot (i.e. fourth, fifth, ninth column with bottom label KPI2) the values are always less or equal than the established 25 and therefore KPI2 is always satisfied. However, the number of allowed requests for PhD Students are always between 8 and 19, with an average of 12, meaning a quite good resource allocation. While, for Master Students and Professors the situation is quite different. Allowed booking requests for Master Students varies from 0 to 4 with an average of 1.2; for Professors varies from 0 to 2 with an average of 0.4. This means that in the morning and evening time slot the Multimedia Laboratory is almost empty. This is in line with the results obtained for KPI1.1, KPI1.2 and KPI1.3.

The KPIs analysis stressed the exigence of an access control policy improvement specifically for the Master Students and Professors. Suggestions come from a detailed analysis of the requests covering the Rule Entity RE_4, i.e. the requests denied by the PDP. In collecting coverage and KPIs data, the Policy Monitoring Infrastructure provides to the Policy Validator additional information about the nature of the denied requests. Specifically the Policy Monitoring Infrastructure evidences which kind of subject is booking the Multimedia Laboratory and at what time. These data are reported in Table 3 in the second and third column for Master Students, in the sixth and seventh column for Professors, and in the eighth and tenth column for PhD Students. Analyzing this information the Policy Validator can observe that most of the requests from Master Students are associated to the time slot 17–21, while few variations in booking activity can be observed for the Professors (just very small increase for the 9–13 time slot).

An immediate possible improvement is therefore to swap the time slots implemented in the access control policy for Master Students and Professors i.e. changing in the first rule (ruleA) at lines 47 from 9 to 17, and in the third rule (ruleC) at line 117 from 17 to 9.

Table 3. Experiment results: first round

	Master Student			Professor			PhD Student						
Day	17-21	13-17	9-13	17-21	13-17	9-13	17-21	13-17	9-13	% KPI1	% KPI1.1	% KPI1.2	% KPI1.3
1	14	12	2	0	0	0	2	14	2	34.78	4.35	0	30.43
2	17	10	1	2	2	0	2	12	2	31.25	2.08	4.17	25
3	10	14	3	0	0	0	4	14	0	37.78	6.67	0	31.11
4	9	11	0	0	0	2	3	18	2	40	0	0	40
5	17	10	0	2	2	0	1	15	2	34.7	0	4.08	30.61
6	18	5	0	0	0	0	6	15	2	32.61	0	0	32.61
7	21	6	1	0	0	0	1	19	1	40.82	2.04	0	38.78
8	13	12	0	0	0	1	2	11	4	25.58	0	0	25.58
9	23	10	1	2	2	0	4	8	0	22	2	4	16
10	20	9	3	0	0	2	2	9	1	26.09	6.52	0	19.57
11	17	6	2	2	2	0	5	15	0	38.78	4.08	4.08	30.61
12	16	11	0	1	1	1	4	11	1	26.09	0	2.17	23.91
13	14	8	2	0	0	3	4	15	1	36.17	4.26	0	31.91
14	23	8	1	0	0	1	2	11	1	25.53	2.13	0	23.40
15	19	10	1	0	0	1	3	9	2	22.22	2.22	0	20
16	14	14	1	1	1	2	3	10	2	25	2.08	2.08	20.83
17	22	10	1	0	0	0	5	10	0	22.92	2.08	0	20.83
18	17	12	1	0	0	0	2	12	2	28.26	2.17	0	26.09
19	18	14	0	0	0	1	4	6	2	13.33	0	0	13.33
20	22	9	1	0	0	2	3	9	1	21.28	2.13	0	19.15
21	23	8	4	0	0	0	5	9	1	26	8	0	18
22	12	17	1	0	0	2	1	8	2	20.93	2.33	0	18.60
23	17	11	2	0	0	1	1	13	1	32.61	4.35	0	28.26
24	15	13	0	0	0	0	0	18	0	39.13	0	0	39.13
25	14	19	2	0	0	0	2	9	2	22.92	4.17	0	18.75
average	17	10.76	1.2	0.4	0.4	0.76	2.84	12	1.36	29.07	2.55	0.82	25.70
			KPI2	KPI2				KPI2					

For aim of completeness we performed a second simulation experiment using the same setting of the previous one but with the access control policy modified as described above. In Table 3 we reported the obtained results.

From the analysis of the new results the Policy Validator could observe that:

– Excluding 6 observation windows, where the coverage percentage is in any case greater than 54%, the KPI1 is almost satisfied with an average value of

Table 4. Experiment results: second round

	Master Student			Professor			PhD Student						
Day	17-21	13-17	9-13	17-21	13-17	9-13	17-21	13-17	9-13	% KPI1	% KPI1.1	% KPI1.2	% KPI1.3
1	21	9	3	0	0	0	2	12	1	68.75	43.75	0	25
2	20	14	1	0	0	0	3	8	0	60.87	43.48	0	17.39
3	20	9	0	0	0	0	2	14	4	69.39	40.82	0	28.57
4	16	13	1	0	0	0	4	10	3	55.32	34.04	0	21.28
5	17	11	1	0	0	0	2	9	2	61.91	40.48	0	21.43
6	16	11	1	0	0	2	2	13	2	65.96	34.04	4.26	27.66
7	16	13	4	0	0	0	0	11	1	60	35.56	0	24.44
8	15	8	3	0	0	2	4	12	1	64.44	33.33	4.44	26.67
9	18	12	3	1	1	0	1	9	2	57.45	38.30	0	19.15
10	15	9	0	0	0	2	2	17	1	73.91	32.61	4.35	36.96
11	19	11	0	0	0	2	1	14	1	72.92	39.58	4.17	29.17
12	24	9	2	0	0	1	3	6	2	65.96	51.06	12.13	12.77
13	16	11	0	0	0	0	3	12	1	65.12	37.21	0	27.91
14	12	8	2	1	1	0	7	14	1	56.52	26.09	0	30.44
15	21	7	0	1	1	0	2	16	1	75.51	42.86	0	32.65
16	17	13	2	0	0	1	1	15	0	67.35	34.69	20.04	30.61
18	20	10	1	0	0	1	2	11	2	68.10	42.55	2.13	23.40
19	18	10	2	0	0	3	5	5	1	59.09	40.91	6.82	11.36
20	18	10	2	0	0	2	1	12	1	69.57	39.13	4.35	26.09
21	18	14	1	0	0	1	5	7	2	54.17	37.05	2.08	14.58
22	19	9	1	0	0	0	5	12	2	64.58	39.58	0	25
23	18	9	2	0	0	0	0	11	3	67.44	41.86	0	25.58
24	20	8	2	1	1	0	4	12	0	66.67	41.67	0	25
25	17	13	2	0	0	0	5	7	0	54.55	38.64	0	15.91
average	18.12	10.32	1.44	0.16	0.16	0.76	2.72	11.08	1.44	64.80	39.24	1.66	23.91
	KPI2					KPI2		KPI2					

64.80%. The implemented changes in the access control policy provide a big improvement to this KPI and confirm that now the access control policy is more close to the real booking behavior of different users (Table 4).

– From the detailed analysis of the data collected for KPI1.1, KPI1.2 and KPI1.3 emerges that: KPI1.3 is still most of the times satisfied (a part from 5 observation windows where the percentage of coverage in any case greater than 11%); percentage of accepted booking requests from Master Students is increased on average (average value is 39.24%) and only in 7 observation windows the value is less than the boundary 36%, but in any case greater than 26%. The KPI1.2 is still not satisfied with values low than the established 6%. In such specific case, by analyzing the data of Professors booking requests (see fiftht and seventh column) it is evident that in any case Professors rarely ask for booking Multimedia Laboratory. To improve this situation alternative way of modifying the access policy should be considered.
– The daily number of allowed requests in each time slot (i.e. second, seventh and ninth column with bottom label KPI2) are still always less or equal than 25 and therefore KPI2 always satisfied. However, the number of allowed requests for Master Students is improved a lot with values varying from 12 to 24 with an average of 18,12. This means a good improvement for the resource

allocation for Master and PhD Students even if the situation for Professors remain almost unchanged.

Of course further improvements to the access control policy could be possible in particular for improving the resource allocation during the 9-13 slot where the Multimedia Laboratory remains almost empty. For instance providing parallel booking slot sessions and so on. However this would be a simple experiment to illustrate the importance and the potentialities of a Policy Monitoring Infrastructure inside an access control systems. Indeed without such a feature the analyzed data were not available and improvements not easily identifiable.

5 Related Works

This work spans over several research directions, including: monitoring of distributed systems and access control system validation.

5.1 Access-Control

In the last decades a large variety of policies and policy models have been proposed to define authorized operations on specific resources. Well known examples of access control models include Mandatory Access Control (MAC) Model, Discretionary Access Control (DAC) Model, Role-Based Access Control (RBAC) Model and more recently eXtensible Access Control Markup Language (XACML) conforming services and applications. Many framework have been presented for specification and validation of access control policies [13–15]. Some works such as [13] use UML based models for expressing role based access control on domain concepts and provide automated facilities for translating these models into XACML code. Other works extract information from business process models to formulate a set of XACML based security policies [16] or derive process-related RBAC models from process execution histories [17]. These models do not consider policy detection and management mechanisms in large and complex environments able to detect inconsistencies of policy specifications and allow policy adaptability to evolving contextual behaviour or technological environment. Some proposals representing an attempt to address this issue are the works in [6,18]. Specifically, the authors of [6] present a dynamic policy management process and access management system that integrates the analysis of user management data as well as contextual data. Similarly to our approach, this work addresses policy recommendations based on contextual data and KPI validation but it relies on mining engine more than on adaptable and flexible monitoring facilities. The work in [18] presents a self adaptive authorization framework to automatically identifying security flaws and autonomously change the access control policy configuration. Similarly to our approach this framework uses monitoring facilities of the authorization system to identify abnormal behaviour and is compliant with RBAC/ABAC authorization infrastructure. Differently from [18], the main goal of the policy monitoring infrastructure proposed in this paper is not to implement self adaptation but providing feedbacks to policy validators for updating and refining policies according to contextual changes.

5.2 Monitoring

Several general-purpose monitoring proposals are currently available, which can be mainly divided into two groups: those that are embedded in the execution engine such as [19,20] and those that can be integrated into the execution framework as an additional component such for instance [21]. Both the solutions have specific advantages. For sure, an embedded solution reduces the performance delay of the execution framework, mainly in terms of interactions and communication time. Coverage indicators can be directly evaluated by the execution framework, which can also execute corrective actions in case of important deviations. The main disadvantage of these approaches is the lack of flexibility both in the data collection, coverage measures definition and language adopted. Usually, in these proposals all the interested parameters, have to be predefined and modeled directly into the execution engine, by means of specific editors, and are dependent on the notation used for the policy definition. Thus any change requires to redesign or improve the execution engine itself, preventing in such manner the possibility of dynamic modification.

Among the additional monitor facility in this paper we refer to the monitoring framework called Glimpse [12], which is extremely flexible and adaptable to various scenarios and SOA architecture patterns.

6 Discussion and Conclusions

In this paper we proposed a Policy Monitoring Infrastructure to dynamically validate and monitor the actual resources access and users behavior in order to update, correct or improve the policy specification in granting/denying the accesses. Through the proposal of this paper policy validators can have a better knowledge of the actual policy usage, define and on-line calculate possible key performance indicators (KPIs) useful to adjust the policies and detect possible policy vulnerabilities or outdated rules.

The results collected in the experiment demonstrated the feasibility of the proposal and highlighted some of the potentialities of the Policy Monitor Infrastructure. Indeed without such a feature most of the weaknesses evidenced in the experiment and suggestions for access control policy improvements were not easily identifiable.

In its simplicity the experiment open the path for further improvements of the proposed infrastructure. As future work we would like to integrate in the Infrastructure other coverage criteria, which can be more focused on the peculiarities of the access control system adopted in the different industrial context. We would like also to extend the Policy Monitoring Infrastructure considering different access and usage control policy specification languages. Finally we are planning to include in the Infrastructure a component able to automatically infer, from the coverage data collected, an alternative operational access control policy so to automatically validate and assess the original the XACML policy against the modification or changes experienced in the real world usage. This can help the policy validators to get a clearer idea of the constraints and permissions expressed in the policy and identify the possible improvements or modifications.

Concerning threats to validity of the presented experiment, four aspects can be considered: the Rule Entities generation, user profiling for the simulation set up, the KPIs specification and the derivation of the XACML requests. Indeed, the coverage criterion, the sample of users selected during the interview stage and the KPIs defined may have influenced the reported results. Moreover due to manually derived XACML requests as well as the random selection of test cases in each observation windows, could be that different choices might have provided different effectiveness results. However the experiment would only be an example of Policy Monitoring Infrastructure usage.

Acknowledgments. This work has been partially supported by the GAUSS national research project (MIUR, PRIN 2015, Contract 2015KWREMX).

References

1. OASIS: Extensible Access Control Markup Language (XACML) version 2.0, February 2005
2. Martin, E., Xie, T., Yu, T.: Defining and measuring policy coverage in testing access control policies. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 139–158. Springer, Heidelberg (2006). https://doi.org/10.1007/11935308_11
3. Bertolino, A., Daoudagh, S., El Kateb, D., Henard, C., Le Traon, Y., Lonetti, F., Marchetti, E., Mouelhi, T., Papadakis, M.: Similarity testing for access control. Inf. Softw. Technol. **58**, 355–372 (2015)
4. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E., Schilders, L.: Automated testing of extensible access control markup language-based access control systems. IET Softw. **7**(4), 203–212 (2013)
5. Calabrò, A., Lonetti, F., Marchetti, E.: Access control policy coverage assessment through monitoring. In: Proceedings of TELERISE 2017 Workshops, Trento, Italy, 12 September 2017 (2017, to apper)
6. Hummer, M., Kunz, M., Netter, M., Fuchs, L., Pernul, G.: Adaptive identity and access management–contextual data based policies. EURASIP J. Inf. Secur. **2016**(1), 19 (2016)
7. Rapps, S., Weyuker, E.: Selecting software test data using data flow information. IEEE Trans. Softw. Eng. **SE-11**(4), 367–375 (1985)
8. Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. ACM Comput. Surv. **29**(4), 366–427 (1997)
9. Bertolino, A., Marchetti, E., Morichetta, A.: Adequate monitoring of service compositions. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2013, pp. 59–69 (2013)
10. Bertolino, A., Calabró, A., Lonetti, F., Marchetti, E.: Towards business process execution adequacy criteria. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2016. LNBIP, vol. 238, pp. 37–48. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27033-3_3
11. Bertolino, A., Le Traon, Y., Lonetti, F., Marchetti, E., Mouelhi, T.: Coverage-based test cases selection for XACML policies. In: Proceedings of ICST Workshops, pp. 12–21 (2014)

12. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a model-driven infrastructure for runtime monitoring. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 130–144. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24124-6_13
13. Daoudagh, S., El Kateb, D., Lonetti, F., Marchetti, E., Mouelhi, T.: A toolchain for model-based design and testing of access control systems. In: 2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 411–418. IEEE (2015)
14. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E.: An automated testing framework of model-driven tools for XACML policy specification. In: 2014 9th International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 75–84. IEEE (2014)
15. Ferraiolo, D., Atluri, V., Gavrila, S.: The policy machine: a novel architecture and framework for access control policy specification and enforcement. J. Syst. Architect. **57**(4), 412–424 (2011)
16. Wolter, C., Schaad, A., Meinel, C.: Deriving XACML policies from business process models. In: Weske, M., Hacid, M.-S., Godart, C. (eds.) WISE 2007. LNCS, vol. 4832, pp. 142–153. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77010-7_15
17. Baumgrass, A., Schefer-Wenzl, S., Strembeck, M.: Deriving process-related RBAC models from process execution histories. In: 2012 IEEE 36th Annual Computer Software and Applications Conference Workshops, pp. 421–426, July 2012
18. Bailey, C., Chadwick, D.W., De Lemos, R.: Self-adaptive authorization framework for policy based RBAC/ABAC models. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), pp. 37–44. IEEE (2011)
19. Daoudagh, S., Lonetti, F., Marchetti, E.: Assessment of access control systems using mutation testing. In: Proceedings of TELERISE, pp. 8–13 (2015)
20. Mouelhi, T., El Kateb, D., Le Traon, Y.: Chapter five-inroads in testing access control. Adv. Comput. **99**, 195–222 (2015)
21. Carvallo, P., Cavalli, A.R., Mallouli, W., Rios, E.: Multi-cloud applications security monitoring. In: Au, M.H.A., Castiglione, A., Choo, K.-K.R., Palmieri, F., Li, K.-C. (eds.) GPC 2017. LNCS, vol. 10232, pp. 748–758. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57186-7_54

Requirements Engineering and Requirements-Based Testing

On Evidence-Based Risk Management in Requirements Engineering

Daniel Méndez Fernández[1](✉), Michaela Tießler[1], Marcos Kalinowski[2], Michael Felderer[3], and Marco Kuhrmann[4]

[1] Technical University of Munich, Munich, Germany
{Daniel.Mendez,Michaela.Tiessler}@tum.de
[2] Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
kalinowski@inf.puc-rio.br
[3] University of Innsbruck, Innsbruck, Austria
michael.felderer@uibk.ac.at
[4] Clausthal University of Technology, Clausthal-Zellerfeld, Germany
kuhrmann@acm.org

Abstract. Background: The sensitivity of Requirements Engineering (RE) to the context makes it difficult to efficiently control problems therein, thus, hampering an effective risk management devoted to allow for early corrective or even preventive measures.
Problem: There is still little empirical knowledge about context-specific RE phenomena which would be necessary for an effective context-sensitive risk management in RE.
Goal: We propose and validate an evidence-based approach to assess risks in RE using cross-company data about problems, causes and effects.
Research Method: We use survey data from 228 companies and build a probabilistic network that supports the forecast of context-specific RE phenomena. We implement this approach using spreadsheets to support a light-weight risk assessment.
Results: Our results from an initial validation in 6 companies strengthen our confidence that the approach increases the awareness for individual risk factors in RE, and the feedback further allows for disseminating our approach into practice.

Keywords: Requirements engineering · Risk management
Evidence-based research

1 Introduction

Requirements Engineering (RE) has received much attention in research and practice due to its importance to software project success. Precise and consistent requirements directly contribute to appropriateness and cost-effectiveness in the development of a system [1] whereby RE is a critical determinant of productivity and software (process) quality [2]. Yet, RE remains an inherently complex discipline due to the various individual socio-economic and process-related influences in the industrial environments. In contrast to many other disciplines, RE is largely characterised by the particularities of the application domain, and it is largely driven by uncertainty and also by human factors. All this influences the way of working in individual contexts including the choice of methods, approaches, and tools in RE. In fact, what might work very well in one project setting might be completely alien to the needs and the culture of the next [3]. The lack of possibilities to standardise the discipline with universal solutions renders it consequently difficult to efficiently control and optimise the quality of RE as an appropriate means to an end.

In consequence, the individual particularities of the contexts hamper an effective risk management devoted to allow for early corrective or even preventive measures. Hence, it is not surprising that despite the importance of excellence in RE, we can still observe various problems in industrial environments rooted all in insufficient RE [4,5]. Requirements Engineering risk factors which might be typical for one project setting, might not be valid for others; for instance, while one project might be characterised by frequent changes in the requirements, the requirements in another project might be stable. Another factor to be considered is the currently weak state of empirical evidence in RE [4,5]. The lack of proper empirical figures that would demonstrate what phenomena occur in practical settings, what problems practitioners face, and what success factors we can infer for the various contexts makes it impossible for available risk management approaches to consider context-sensitive risk factors and related corrective and even preventive measures for RE.

To efficiently consider the particularities of RE in risk management approaches, we postulate that we first need knowledge about the current state of practice in industrial environments and problems faced therein. Motivated by this situation, we initiated the *Naming the Pain in Requirements Engineering* (NaPiRE) initiative (see Sect. 2). NaPiRE constitutes a globally distributed family of bi-yearly replicated surveys on the industrial status quo and problems in RE. This allows to steer research in RE in a problem-driven manner and the establishment of proper solutions based on a better understanding of the needs of the various industrial contexts.

In this paper, we contribute the first steps towards a holistic evidence-based RE risk management approach that explicitly takes into account the particularities in RE as described above. We concentrate on assessing risk factors in RE. In particular, we use the cross-company data from NaPiRE to build a holistic model of context-sensitive problems, causes, and effects. We use this knowledge to implement a probabilistic network that allows to calculate the posterior probability of certain risk factors based on knowledge about the current situation. We use this to propose a first evidence-based RE risk assessment approach which we validate in 6 companies. We conclude by outlining current work on integrating guidelines to mitigate and correct problems in RE to make the consequential next step from a RE risk assessment approach to a holistic, evidence-based RE risk management approach.

Our contribution and the feedback by the practitioners supports us and other researchers already in proposing further solutions to requirements engineering that need to rely on cross-company data. Further, our proposed risk assessment approach already allows practitioners to reflect on their current situation by increasing their awareness for potential problems in RE.

The remainder of this paper is organised as follows: In Sect. 2 we introduce the background and related work. In Sect. 3, we elaborate on the research design. In Sect. 4, we describe our overall risk assessment approach and in Sect. 5, we report on a validation conducted in 6 companies, before concluding our paper in Sect. 6.

2 Background and Related Work

Software risk management constitutes means to efficiently assess and control risk factors affecting the overall software development activities [6] (planned ones and deviations) and is often associated with "project management for adults" as baptised by DeMarco et al. [7]. The importance of risk management for software engineering in general and requirements engineering in particular has been addressed in several risk management approaches tailored to software engineering processes [8]. Already the spiral model of Boehm [9] explicitly includes risk management within software development. The Riskit method [10] provides a sound theoretical foundation of risk management with a focus on the qualitative understanding of risks before their possible quantification. Karolak [11] proposes a risk management process for software engineering that contains the activities risk identification, risk strategy, risk assessment, risk mitigation, and risk prediction. With ISO 31000 [12], which was released in 2009, there is even a family of standards for risk management available that can also be instantiated in software engineering and its subareas like testing, where ISO 31000 has already been applied in the context of risk-based testing [13], or requirements engineering.

A recent study on industrial development practices [14] shows that practitioners see the need to explicitly include traditional risk management approaches into RE that tends to be done rather agile. This further strengthens our confidence in the need to tailor risk management approaches to the particularities of RE actively taking into account the volatility therein (as discussed in our introduction).

In fact, most work on risk management in the context of requirements engineering focuses on identifying risks in a bottom-up approach and analysing risks during the requirements engineering process. For instance, Asnar et al. [15] provide a goal-driven approach for risk assessment in requirements engineering, and Haisjackl et al. [16] provide an estimation approach and respective tool support [16]. For risk management within requirements engineering itself, Lawrence et al. [17] provide a list of top risks in requirements engineering, which includes overlooking a crucial requirement, inadequate customer representation, modelling only functional requirements, not inspecting requirements, attempting to perfect requirements before beginning construction as well as representing requirements in the form of designs. However, evidence-based approaches to risk management in requirements engineering as proposed in this paper are so far not available. For such an approach, we need a proper empirical basis on requirements engineering which has been recently established by the NaPiRE (Naming the Pain in Requirements Engineering) initiative.

The NaPiRE initiative was started in 2012 in response to the lack of proper empirical figures in RE research. The idea was to establish a broad survey investigating the status quo of RE in practices together with contemporary problems practitioners encounter. This should lead to the identification of interesting further research areas as well as success factors for RE. We created NaPiRE as a means to collaborate with researchers from all over the world to conduct the survey in different countries. This allows us to investigate RE in various cultural environments and increase the overall sample size. Furthermore, we decided to run the survey every two years so that we can cover slightly different areas over time and have the possibility to observe trends. NaPiRE aims to be open, transparent and anonymous while yielding accurate and valid results.

At present, the NaPiRE initiative has over 50 members from more than 20 countries mostly from Europe, North-America, South-America, and Asia. There have been two runs of the survey so far. The first was the test run performed only in Germany and in the Netherlands in 2012/13. The second run was performed in 10 countries in 2014/15. All up-to-date information on NaPiRE together with links to instruments used, the data, and all publications is available on the web site http://www.re-survey.org. The first run in Germany together with the overall study design was published in [4]. It already covered the spectrum of status quo and problems. Overall, we were able to get full responses from 58 companies to test a proposed theory on the status quo in RE. We also made a detailed qualitative analysis of the experienced problems and how they manifest themselves. For the second run, we have published several papers [18–21] concentrating on specific aspects and the data from only one or two countries and one paper [5] focusing on RE problems, causes and effects based on the complete data set covering data reported by 228 companies. An analysis of the data with a focus on risk management and evidence-based risk management in RE has not been published so far.

3 Research Design

Our overall goal is to provide first steps towards a holistic evidence-based RE risk management approach that allows to steer a context-sensitive risk management based on empirical cross-company data and, thus, bridges shortcomings of currently available approaches. Our research, therefore, needs to rely on data reflecting practical problems in RE that are typical for certain context factors. The scope of validity (and relevance) of the proposed solution consequently depends on the practical contexts from which the data was obtained and where we applied our approach.

Our research design is therefore strongly inspired by the (design science) engineering cycle as exemplary described by Wieringa and Moralı in [22]. That is, we follow a cyclic development approach which is initially based on idealised assumptions, solution design, and validation as well as evaluation in practice, before revising our assumptions. With each iteration, we can make more realistic assumptions and scale our solution proposal and its effects up to practice. However, in contrast to classical design science research aimed at practical problem solving, where we develop individual solutions to practical problems, our aim is not to solely understand the effects of such solution proposals. Instead, our aim is also to better understand the particularities of contexts and phenomena involved and to actively incorporate them in our solution. That is, we begin with understanding which problems practitioners experience in certain contexts, develop our evidence-based RE Risk Assessment approach and – by transfer into practice – we can use the observations to make our context assumptions more reliable and precise for the next iteration.

The resulting methodological approach reflects the basic notion of knowledge transfer. Our model is therefore structured in analogy to the technology transfer model as described by Gorschek et al. [23] and sketched in Fig. 1 to visualise the cyclic nature.

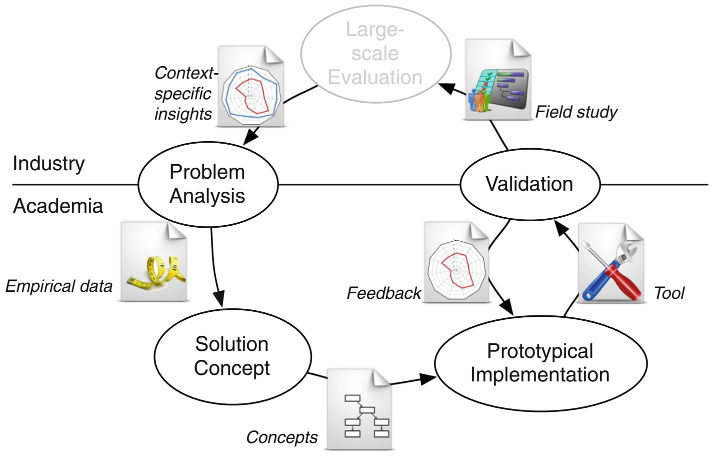


Fig. 1. Overall research design. The last step including a field study as part of a large-scale evaluation is in scope of current activities and out of scope of this paper.

Starting with a problem analysis based on cross-company data emerging from the NaPiRE initiative, we develop a first solution proposal, which we transfer into a prototypical tool implementation. We use this prototype for a first validation with industry participants as a preparation for a large-scale evaluation. At the end of the iteration, we use the observations from the large-scale evaluation to increase our empirical data set with additional context factors before entering the next development iteration.

In this paper, we report on the first iteration where we develop and validate our initial approach to evidence-based RE Risk Assessment and outline the next steps of a continuous evaluation and refinement towards a holistic Risk Management approach which is in scope of current work.

In the following, we introduce the single steps to the extent necessary in context of our contribution at hands.

3.1 Problem Analysis

Our data for the initial problem analysis is largely based on the last replication of the NaPiRE initiative (see Sect. 2), including cross-company data from 228 companies located in 10 countries. There, we investigated the status quo on the practices applied in requirements engineering as well as problems, causes, and effects as experienced and reported by practitioners from various domains. Table 1 exemplarily illustrates the most frequently cited top 10 problems as reported by the respondents starting with an accumulation of the problems and the summary of the single (top 5) ranks of the problems.

Table 1. Most cited top 10 RE problems as reported in [5].

RE Problem	Total	Cause for project failure	Ranked as #1	Ranked as #2	Ranked as #3	Ranked as #4	Ranked as #5
Incomplete and/or hidden requirements	109 (48%)	43	34	25	23	17	10
Communication flaws between project team and customer	93 (41%)	45	36	22	15	9	11
Moving targets (changing goals, business processes and/or requirements)	76 (33%)	39	23	16	13	12	12
Underspecified requirements that are too abstract	76 (33%)	28	10	17	18	19	12
Time boxing/not enough time in general	72 (32%)	24	16	11	14	17	14
Communication flaws within the project team	62 (27%)	25	19	13	11	9	10
Stakeholders with difficulties in separating requirements from known solution designs	56 (25%)	10	13	13	12	9	9
Insufficient support by customer	45 (20%)	24	6	13	12	6	8
Inconsistent requirements	44 (19%)	15	8	9	6	9	12
Weak access to customer needs and/or business information	42 (18%)	16	7	10	8	8	9

We can see, for instance, that incomplete requirements dominate the list of top 10 problems in RE directly followed by communication flaws between the project team and the customer. To elaborate on risk factors and how they propagate in a project ecosystem, we need, however, knowledge going beyond single problems.

From Problems to Cause-Effect Chains. For each of the problems reported, we further analyse the data on the causes and the effects to increase our understanding about the criticality of the problems and their root causes. The latter is important to understand possibilities for corrective or even preventive measures. For instance, the main causes for communication flaws between the project team and the customer are of organisational nature, including causes such as language barriers, missing engagement by the customer, or missing direct communication with the customer. Effects of the problem include, inter alia, a poorer product quality by incorrect or missing features. A richer introduction into the problems, causes, and effects in RE can be taken from our previously published material [5].

Figure 2 illustrates the full-scale model of the causes and the effects of the problems as distilled from the NaPiRE data set. Visualising the full-scale model serves two purposes: It shall demonstrate that (1) the model includes complex cross-dependencies and single phenomena can lead to several problems, and that (2) despite the complexity of the model, it already supports a basic understanding of the phenomena involved in software development projects and how they propagate through the different phases. Yet, such a model still does not help understanding context-sensitive conditional dependencies which renders it nearly useless for operationalisation as a risk assessment approach. That is, for an effective evidence-based RE risk assessment, we need data that allows us to infer, at least with a certain level of confidence, those effects certain problems have based on a selection of specific context-specific conditions that characterise a project situation at any point in time.

From Cause-Effect Chains to Conditional Probabilistic Distributions. Finally, to lay the foundation for an evidence-based risk assessment which includes operationalisable conditional dependencies, we analyse the data by making use of Bayesian networks. We already made positive experiences in using Bayesian networks in defect causal analyses [24]. Here, we use Netica [1] to implement the cause-effect information in dependency to context factors such as company size, team distribution or software process model used (agile or plan-driven). Such an implementation allows us to quantify the conditional probabilistic distributions of all phenomena involved. More precisely, it allows us to, based on the NaPiRE data used as learning set, use the Bayesian network inferences to obtain the posterior probabilities of certain phenomena to occur when specific causes are known. Figure 3 shows such an exemplary inference for the consequences and their probabilities of the phenomena *missing direct communication to the customer* (highlighted in grey).

We use this analytically curated data as a basis for the proposal of our evidence-based risk assessment approach.

3.2 Solution Concept and Prototypical Implementation

Once understanding the data and its potential, we elaborate the design of our initial evidence-based RE Risk Assessment approach. This approach shall, in a

[1] https://www.norsys.com/.

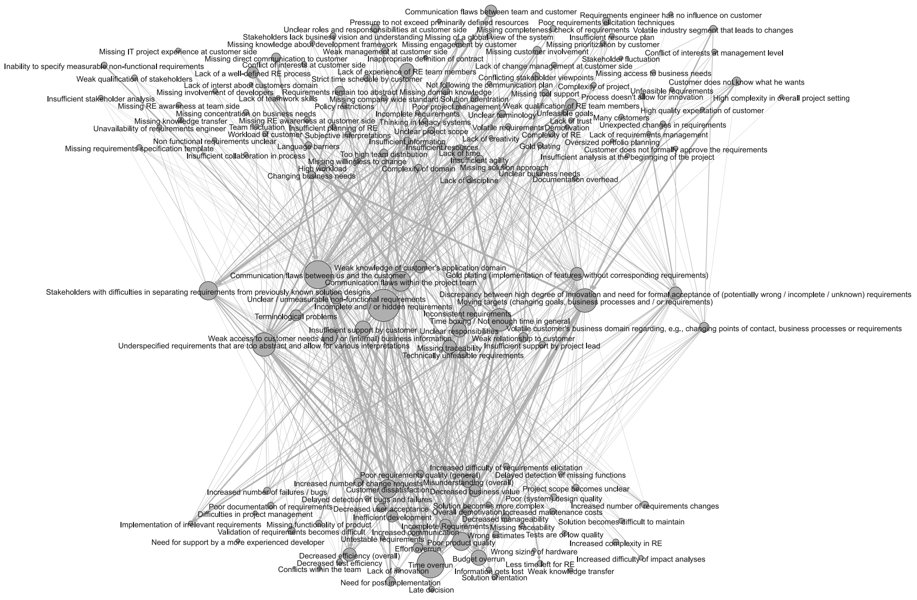

Communication flaws between us and the customer
Weak knowledge of customer's application domain
Communication flaws within the project team
Gold plating (implementation of features without corresponding requirements)
Stakeholders with difficulties in separating requirements from previously known solution designs
Unclear / unmeasurable non-functional requirements
Incomplete and / or hidden requirements
Discrepancy between high degree of innovation and need for formal acceptance of (potentially wrong / incomplete / unknown) requirements
Moving targets (changing goals, business processes and / or requirements)
Terminological problems
Inconsistent requirements
Time boxing / Not enough time in general
Volatile customer's business domain regarding, e.g., changing points of contact, business processes or requirements
Insufficient support by customer
Unclear responsibilities
Weak access to customer needs and / or (internal) business information
Weak relationship to customer
Underspecified requirements that are too abstract and allow for various interpretations
Missing traceability
Insufficient support by project lead
Technically unfeasible requirements

Fig. 2. Full-scale cause-effect visualisation of the RE problems, causes and effects. Not intended to be read in detail (but we cordially invite the reader to zoom in).

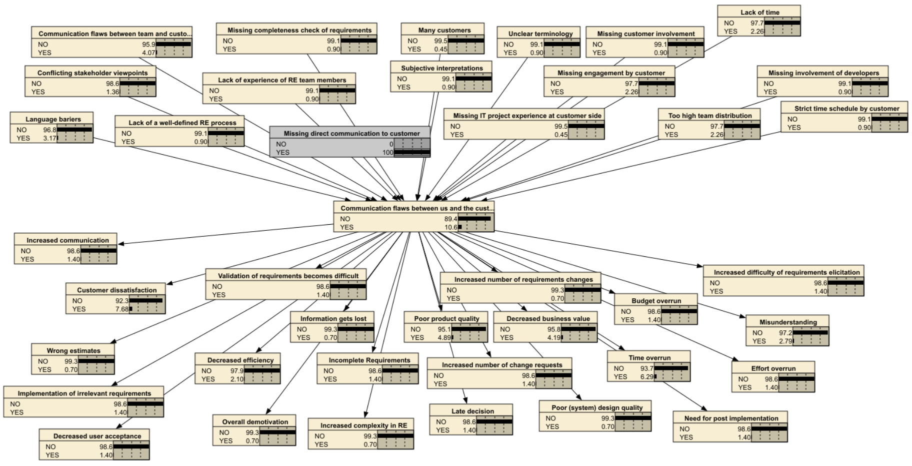

Communication flaws between team and custo... NO 95.9 YES 4.07
Conflicting stakeholder viewpoints NO 98.6 YES 1.36
Language bariers NO 96.8 YES 3.17
Lack of a well-defined RE process NO 99.1 YES 0.90
Missing completeness check of requirements NO 99.1 YES 0.90
Lack of experience of RE team members NO 99.1 YES 0.90
Missing direct communication to customer NO 0 YES 100
Many customers NO 99.5 YES 0.45
Subjective interpretations NO 99.1 YES 0.90
Unclear terminology NO 99.1 YES 0.90
Missing customer involvement NO 99.1 YES 0.90
Lack of time NO 97.7 YES 2.26
Missing engagement by customer NO 97.7 YES 2.26
Missing involvement of developers NO 99.1 YES 0.90
Missing IT project experience at customer side NO 99.5 YES 0.45
Too high team distribution NO 97.7 YES 2.26
Strict time schedule by customer NO 99.1 YES 0.90
Communication flaws between us and the cust... NO 89.4 YES 10.6
Increased communication NO 98.6 YES 1.40
Customer dissatisfaction NO 92.3 YES 7.68
Validation of requirements becomes difficult NO 98.6 YES 1.40
Increased number of requirements changes NO 99.3 YES 0.70
Increased difficulty of requirements elicitation NO 98.6 YES 1.40
Budget overrun NO 98.6 YES 1.40
Misunderstanding NO 97.2 YES 2.79
Wrong estimates NO 99.3 YES 0.70
Information gets lost NO 99.3 YES 0.70
Poor product quality NO 95.1 YES 4.89
Decreased business value NO 95.8 YES 4.19
Time overrun NO 93.7 YES 6.29
Effort overrun NO 98.6 YES 1.40
Decreased efficiency NO 97.9 YES 2.10
Incomplete Requirements NO 98.6 YES 1.40
Increased number of change requests NO 98.6 YES 1.40
Implementation of irrelevant requirements NO 98.6 YES 1.40
Overall demotivation NO 99.3 YES 0.70
Increased complexity in RE NO 99.3 YES 0.70
Late decision NO 98.6 YES 1.40
Poor (system) design quality NO 99.3 YES 0.70
Need for post implementation NO 98.6 YES 1.40
Decreased user acceptance NO 98.6 YES 1.40

Fig. 3. Predictive inference for effects of missing direct communication to the customer.

first step, serve the purpose of providing means to assess risk factors in a light-weight manner based on knowledge about a current project situation – while still offering anchor points for extensions to a holistic risk management approach.

To this end, we conceptualise the elements and constructs necessary for RE risk management via a meta model. This meta model captures the elements necessary for a holistic risk management in RE. Figure 4 visualises a simplified resulting model, while highlighting those elements in scope of the initial approach for a risk assessment, i.e. the approach used for our first iteration presented in this paper. In centre of our attention are events. Events are phenomena that materialise in a measurable manner as well as their causes and effects. Those events either impact tasks carried out in a project or artefacts created/modified in the course of such tasks and can occur with various impacts at a certain probability (i.e. risks).

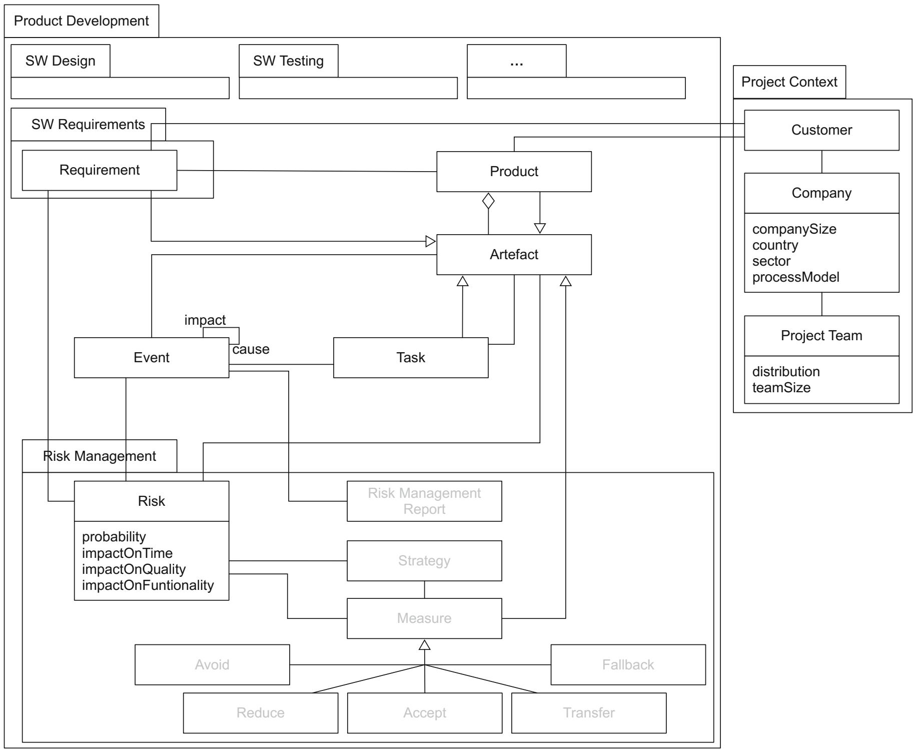


Fig. 4. Strongly simplified meta model for evidence-based RE risk management. Darker elements encompass entities relevant for risk *assessment which* is in scope of first development iteration. Elements in light grey encompass entities necessary for extension to a holistic risk *management* approach.

Of further interest are the context factors used to characterise the development project, such as team characteristics or company characteristics. Please note that the meta model is intentionally incomplete and simplified and shall serve the sole purpose of highlighting those elements in scope of risk assessment. Out of scope, yet necessary for a holistic risk management are the implementation of measures used to, e.g., avoid certain risks.

To apply our approach in practical settings, we implement it via a light-weight prototype. To this end, we rely on spreadsheets realised via MS Excel for the presentation and reporting of risk factors and interactive calculation of the probabilities to encounter those factors based on a current situation (as learnt based on the data analysis described above). We further implement a graphical visualisation of all RE phenomena to increase the awareness of a current project situation using NodeXL. Details on the outcome can be taken from Sect. 4 where we introduce the resulting approach.

3.3 Validation

While the prototypical implementation already serves as an initial proof of concept, we want to better understand the suitability of the approach to practical contexts. This helps us preparing for the large-scale evaluation depicted at the upper part of Fig. 1. To this end, we conduct structured interviews with practitioners in face to face meetings using their answers to potentially revise the proposed (candidate) approach for a broader evaluation. In those interviews, we use the instrument summarised in Table 2 and developed from scratch in a curiosity-driven manner. Further details on the validation can be taken from Sect. 5.

3.4 Large-Scale Evaluation and Next Steps

Once we understand the practitioners' perceptions of our light-weight approach allowing for last modifications of our risk assessment approach, we can scale up to practice by applying it in a longitudinal field study. Based on the outcome of the large-scale evaluation, we intend to gather knowledge about further barriers and limitations in context of risk management in RE. Further, such an evaluation will allow us to further scaling up by integrating proper measures for mitigating potential risks, thus, it allows us for a problem-driven (methodological) transition from risk assessment to risk management. These last steps are in scope of current activities at the time of writing this paper.

3.5 Validity Procedures

It is noteworthy that the proposed approach relies on some assumptions on the NaPiRE data, which could represent threats to validity. The most important one concerns the accuracy of the coding process used to analyse the answers to the open questions RE problems, causes, and effects (used as a basis for the described risk assessment quantification). Coding is essentially a creative task with subjective facets of coders like experience, expertise and expectations.

Table 2. Summary of the interview questions (structure: M = metadata, A = approach, T = tool; type: FT = open free text, SC = closed single choice, LI = Likert scale rating).

	ID	Question	Type
M	1	What is the size of your company?	FT
M	2	Please briefly describe the main sector of your company and the application domain of the software you build	FT
M	3	How would you characterise the projects that you typically work in?	FT
M	4	What is your role in those projects?	FT
M	5a	Is there already a methodology to manage risks in your company?	SC
M	5b	How do you manage risks in your company?	FT
M	6	Which characteristic values would you consider for risk assessment?	FT
M	7	Does this risk management also address risks in Requirements Engineering?	FT
M	8	How important is it to address RE Risks in Risk Management?	LI
M	9	Do you think it makes sense to distinguish RE Risk Management from the overall Risk Management?	LI
A	10	Do you think the approach is more precise than other risk assessment approaches?	SC
A	11a	Do you think it makes sense to assess the risks based on phenomena?	SC
A	11b	Why?	FT
A	12	What are the benefits the approach?	FT
A	13	What do you dislike about the approach?	FT
A	14	What are the barriers for the approach?	FT
A	15	What should we do to improve the approach?	FT
T	16	How appropriate do you think of criticality as a parameter for risk assessment?	LI
T	17	Do you think using the tool would improve your performance in doing your job?	LI
T	18	Do you think using the tool at work would improve your productivity?	LI
T	19	Do you think using the tool would enhance your effectiveness in your job?	LI
T	20	Would you find the tool useful in your job?	LI
T	21	Do you think learning to operate the tool would be easy for you?	LI
T	22	Would you find it easy to get the tool to do what you want to do?	LI
T	23	Do you think it would be easy for you to become skilful in the use of the tool?	LI
T	24	Do you find the tool easy to use?	LI
T	25a	Would you use such a tool for RE risk assessment?	LI
T	25b	Why?	FT
T	26	What would be necessary for using the tool?	LI
T	27a	How would you rate the tool in total?	LI
T	27b	Why?	FT
T	28	How would you rate the risk assessments provided by the tool in total?	LI
T	29	Would you rather adopt a pre-implemented tool or implement it with the risks that applied in your company?	SC
	30	Do you have further remarks?	FT

This threat was minimised by peer-reviewing the coding process. Concerning the NaPiRE survey itself, it was built on the basis of a theory induced from available studies and went through several validation cycles and pilot trials. Further details on the research methods employed can be taken from our previous publication [5].

4 Evidence-Based RE Risk Assessment Approach

This section presents the synthesised evidence-based risk management approach. Section 4.1 introduces into the overall approach from a conceptual level, and Sect. 4.2 presents the prototypical implementation.

4.1 Overall Approach to Evidence-Based RE Risk Assessment

The risk assessment is integrated with an overall project-based risk management. Figure 5 provides the approach's concept. Right in a *Project*, we integrate the RE risk assessment with the overall project's risk management and expect a continuous implementation, i.e. we assume that a company implements its own risk management as a continuous task performed throughout the whole project.

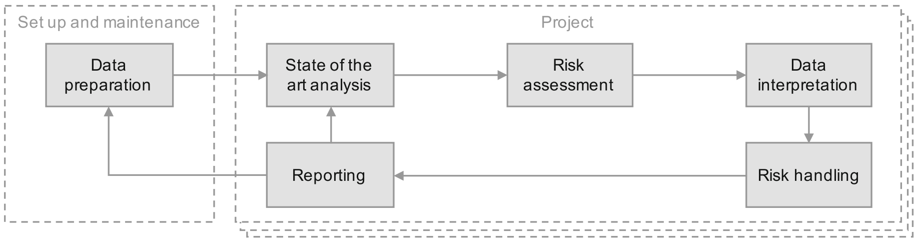


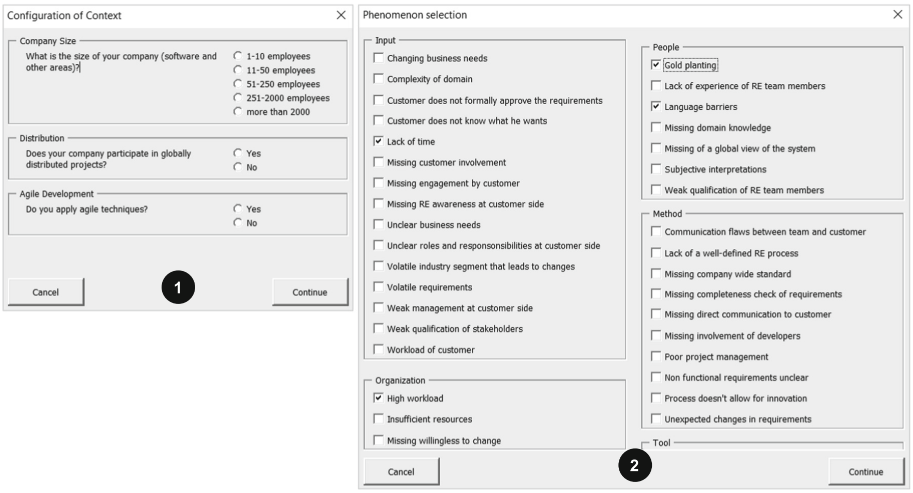
Fig. 5. Elementary concepts of the RE risk assessment approach.

Key to the approach presented is the data used for the (initial) risk assessment (Fig. 5; left). The required data can either taken from an external data pool, in our case NaPiRE data (see Sect. 3), or from historical company data. Eventually, the company has to decide whether or not grounding the risk assessment in internal or external data. Using internal data brings in the opportunity to learn from past projects, i.e. to improve the company-specific risk management approach based on lessons learnt. Using external data instead, however, provides advantages, too. We postulate that by using cross-company data, risks that might have not materialised so far in own project settings can be captured based on third-party experience. Once the data source(s) has been selected, data needs to be prepared in order to carry out the risk assessment. Specifically, cause-effect models need to be developed (e.g., using Netica; Fig. 3) to provide processable input for the analysis tool (Sect. 4.2). In a project, five tasks are carried out:

1. In a state-of-the-art analysis, the current project status is evaluated, possible risks, and already materialising risks are analysed—usually by looking for phenomena that "announce" an upcoming problem.
2. In the assessment, based on the input data, the phenomena and associated risk (cause and effects) are calculated and the impact is estimated.
3. In the interpretation, the risks are analysed and prioritised based on the computed probabilities and expected impact (e.g., using a risk classification matrix as used in PRINCE2 or a self-defined criticality index; see Sect. 4.2).
4. Based on the prioritisation, defined actions to adequately respond to the risks identified are initiated.
5. Eventually, the analysis results and the actions initiated are reported. Reporting is used to complete the cycle and prepare the next assessment iteration, i.e. updated reporting data complement the general data basis. Also, reporting helps to evolve the general data basis, such that further projects can benefit from the reported findings.

4.2 Prototypical Implementation

The overall approach was developed using a variety of tools. Besides the tools used for the analysis, i.e. the modelling of the dependency networks and the probability graphs (Fig. 3, Netica), a prototypical support tool was realised using Microsoft Excel. The tool is applied to a given project setup in two steps. Figure 6 shows the first step—the configuration. In the configuration, the user is asked to characterise the project of interest in terms of the context factors company size, distributed work, and use of agile software development. The second part of the configuration aims at selecting all phenomena (see Sect. 3.1) of interest.



Company Size
What is the size of your company (software and other areas)?
1-10 employees
11-50 employees
51-250 employees
251-2000 employees
more than 2000

Distribution
Does your company participate in globally distributed projects?
Yes
No

Agile Development
Do you apply agile techniques?
Yes
No

Input
- Changing business needs
- Complexity of domain
- Customer does not formally approve the requirements
- Customer does not know what he wants
- [✓ Lack of time]
- Missing customer involvement
- Missing engagement by customer
- Missing RE awareness at customer side
- Unclear business needs
- Unclear roles and responsonsibilities at customer side
- Volatile industry segment that leads to changes
- Volatile requirements
- Weak management at customer side
- Weak qualification of stakeholders
- Workload of customer

Organization
- [✓ High workload]
- Insufficient resources
- Missing willingless to change

People
- [✓ Gold planting]
- Lack of experience of RE team members
- [✓ Language barriers]
- Missing domain knowledge
- Missing of a global view of the system
- Subjective interpretations
- Weak qualification of RE team members

Method
- Communication flaws between team and customer
- Lack of a well-defined RE process
- Missing company wide standard
- Missing completeness check of requirements
- Missing direct communication to customer
- Missing involvement of developers
- Poor project management
- Non functional requirements unclear
- Process doesn't allow for innovation
- Unexpected changes in requirements

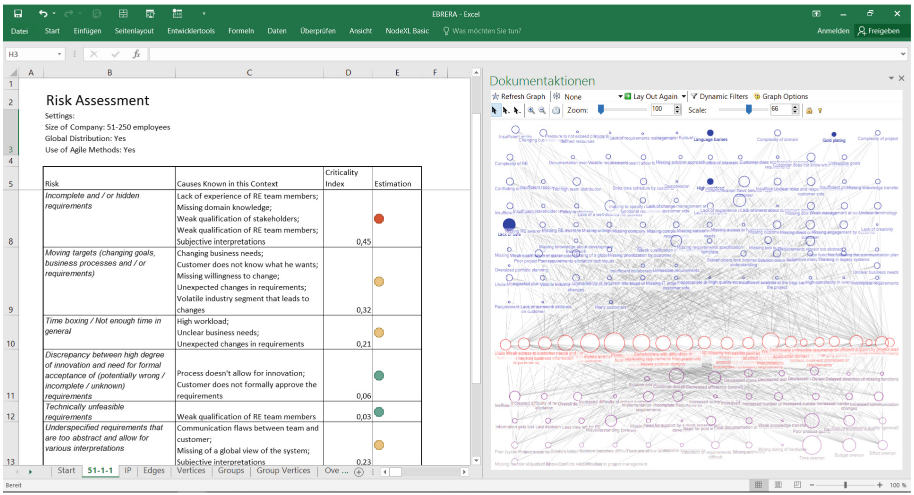
Tool

Fig. 6. Exemplary configuration of the risk assessment tool (basis: NaPiRE data).

Based on the selection, in the second step, the tool generates a report (Fig. 7) that comprises two parts: first, a risk assessment report is generated, which includes the risks and the known context-related causes associated with the phenomena chosen. The assessment report further present the criticality index and an estimate visualising the likelihood of a risk materialising in the current context. In the prototypical implementation, we use a custom criticality index, which is calculated as follows:

$$\frac{p_i}{n} \cdot \frac{p_{ij}}{n_j} \cdot \left(1 + \frac{c_{i_{TRUE}}}{c_i}\right)$$

The index is calculated with the sum of the weighted phenomena c_i, the sum of the weighted phenomena $c_{i_{TRUE}}$ that apply in the current context, the size n of the data set used and the subset n_j used for the phenomena that apply, the frequency of an identified problem in the whole data set p_i, and the frequency of a problem p_{ij} in the subset under consideration.


Risk Assessment

Settings:
Size of Company: 51-250 employees
Global Distribution: Yes
Use of Agile Methods: Yes

Risk	Causes Known in this Context	Criticality Index	Estimation
Incomplete and / or hidden requirements	Lack of experience of RE team members; Missing domain knowledge; Weak qualification of stakeholders; Weak qualification of RE team members; Subjective interpretations	0,45	🔴
Moving targets (changing goals, business processes and / or requirements)	Changing business needs; Customer does not know what he wants; Missing willingness to change; Unexpected changes in requirements; Volatile industry segment that leads to changes	0,32	🟠
Time boxing / Not enough time in general	High workload; Unclear business needs; Unexpected changes in requirements	0,21	🟠
Discrepancy between high degree of innovation and need for formal acceptance of (potentially wrong / incomplete / unknown) requirements	Process doesn't allow for innovation; Customer does not formally approve the requirements	0,06	🟢
Technically unfeasible requirements	Weak qualification of RE team members	0,03	🟢
Underspecified requirements that are too abstract and allow for various interpretations	Communication flaws between team and customer; Missing of a global view of the system; Subjective interpretations	0,23	🟠

Fig. 7. Example of a risk assessment report based on the the configuration from Fig. 6.

The second part of the report visualises the the cause-effect graph in which the selected phenomena are highlighted and embedded in the overall network. This allows for inspecting the phenomena and finding orientation in the complex model.

5 Validation

Our overall objective in our validation is to better understand the current practitioners' views and acceptance of our solution proposal and get first feedback on potential barriers and limitations they see in the presented approach and the proposed tool. This shall help us in preparing for the large-scale evaluation in practice and the dissemination of our solution proposal. To this end, we conduct interviews with 6 practitioners from different companies and formulate the research questions listed in Table 3.

Table 3. Research questions for the interview study.

RQ 1	How are requirements engineering risks assessed in projects?
RQ 2	What are the benefits and drawbacks of the presented approach?
RQ 3	What are the barriers for using the presented approach?
RQ 4	Could practitioners imagine to use the presented tool in practice?
RQ 5	What changes would be necessary to employ it in project settings?

After exploring what the state of practice in risk assessment is in the respective project environments, we are first interested in the general perception of the benefits and drawbacks in the presented approach. To actively prepare for the evaluation and give us the chance for last improvements, we further want to know barriers practitioner see in using our approach, whether they can imagine using the tool, and what changes would be necessary for using our solution proposal. As a preparation for the questions on the tool (RQ 4 and RQ 5), we simulated a given scenario to show the respondents the functionality of the tool in a live demo. The detailed instrument used for the interviews is shown in Sect. 3.3.

Please note that for reasons of confidentiality, but also because the answers were all recorded in German, we have to refrain from disclosing the raw data. In the following, we briefly introduce into the demographics before addressing each of the presented research questions.

5.1 Demographics

Our six respondents represent companies which differ in the sector and constraints attached, thus, covering a broader demographical spectrum. All six respondents are situated in Germany and their contextual information are summarised in Table 4.

In the following, we introduce the interview results. Where reasonable and possible, we provide original statements. Please note, however, that these statements are translated from Germany and in parts slightly paraphrased for the sake of understandability.

5.2 RQ 1: State of Practice in RE Risk Assessment

The state of practice in RE risk assessment is as diverse as the respondents' environments. Some have already defined a risk management approach, others

Table 4. Respondents overview (anonymised).

ID	Company size	Main company sector	Role in projects	Project size	Process model	Project distribution
1	300	IT consulting	Business analyst	4–10	Scrum	Nationally distributed
2	100	Quality management, testing	External consultant for RE and Testing	Medium and large	Agile: Scrum, Kanban	Internationally distributed
3	6000–6500	Software development and consulting	Internal consultant	20–40	Scrum, RUP	Internationally distributed
4	20000	Accounting, IT-consulting	IT (Risk)-consultant	N/A	N/A	N/A
5	30000	Insurance	Project manager	2–120	Agile and traditional approaches	Mostly national distributed
6	180000	Software consulting, customer solution development	Requirements engineer	35	Waterfall, agile approaches	Internationally distributed

are currently establishing one. The most mature risk assessment approach we encountered was by respondent 3 who reports that at the beginning of each project, a cost estimation is conducted that includes the assessment of the relative risk of the project by means of a predefined checklist.

As for the characteristics valued most important in context of risk assessment (Q 8), following statement seems most representative for the respondents:

"Probability of the risks and the potential monetary impact (including damages to the reputation)"

When asked if their current risk assessment approach also addresses risks in RE (Q 7), only 2 respondents stated that requirements-related aspects were considered.

As for the importance to address RE risks in Risk Management (Q 8), the respondents had overall an agreement attitude:

"[...] the sooner [risk management] starts, the better" as it enables ?to identify problems at the beginning.?

On a scale from one to five, the respondents from two companies rate it with the highest importance, three with four and only one makes the rating dependent from the contract type. In case of a fixed price contract, he rates it with 4–5; for time and material contracts, he rates the importance with 1–2 only.

5.3 RQ 2: Benefits and Drawbacks of the Approach

When asked on the benefits and drawbacks of the presented approach, the answers provided a rich and diverse picture. 3 of 6 respondents clearly stated that the presented approach would be more precise for RE than available approaches (Q 10) while only 1 stated it would be not. Further, 4 of 6 respondents stated it would make sense to assess risks based on the empirically grounded RE phenomena (Q 11a), while, again, only 1 did not agree. When asked for their rationale (Q 11b), most respondents stated that such an evidence-based view would increase the awareness for risks in RE. A further positive aspect highlighted was that the approach would rely on previously made experiences in same or similar environments:

"Embarrassing to repeat the same mistakes."

Challenges associated with the approach included:

"It makes sense, but it seems difficult to realise in our company" as well as "Projects are not very comparable and, thus, we would need larger amounts of data. Sizing the clusters [sets of context factors] is challenging."

When asked directly for the benefits of the approach (Q 12), our respondents stated that it would offer an experience-based continuous learning including risks one might not be fully aware of, and it would offer the possibility to compare different project situations based on common risk factors. This would improve the situation for project leads by gaining more control.

When asked for what they disliked (Q 13), respondents stated, in one form or the other, that transparency on risk factors experienced by other projects with similar characteristics might lead to more insecurity of the project team:

"Too much data may lead to confusion."

Our respondents further stated that the maintainability of such an approach would constitute another challenge:

"Keeping the data up-to-date is an elaborate task [as we need to] ensure high data quality."

5.4 RQ 3: Barriers for Using the Approach

As the main barriers for using the presented approach (Q 14), our respondents stated, in tune with the previous statement, the need to maintain large amounts of data. A further statement included that a strong focus on risks related to RE might lead to pessimism. A further perceived barrier affected the fear for projects giving up their individual characteristics when being compared with other projects:

> *"Single projects would have the impression their individual characteristics would be neglected."*

When asked what we could do to improve the approach (Q 15), we encountered mainly two aspects: Inclusion of time as a dimension into the phenomena that would allow to calculate the time until risks may manifests as real events, and a better visualisation for especially large amounts of data.

5.5 RQ 4: Usability of the Tool

As for the usability of the tool presented via a live demo, we asked our respondents the questions from Table 2 and visualise the results in Fig. 8. While the focus of our validation was on qualitative feedback, it is still noteworthy that the prototypical implementation yielded overall a reasonably good acceptance while acknowledging that—although the tool offers a light-weight mechanism to assess the risks—it still requires effort for the data maintenance and curation (reflected in a mixed rating in the performance and productivity).

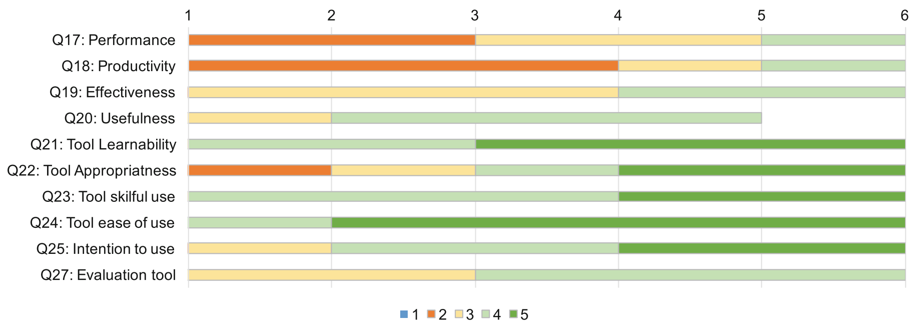


Fig. 8. Evaluation of the concept and the tool prototype with the six practitioners. The figure summarises the number of responses as stacked bar charts, ordered by their agreement on Likert scales from 1 (low/no agreement) to 5 (high agreement).

5.6 RQ 5: Necessary Changes

When asked for the necessary changes we would need to apply to the whole approach and to the corresponding prototypical tool (Q 26), two major aspects were prominent:

> *"Support in the data collection and [possibilities to better] compare company data with the data of other companies" as well as "Improved reliability of the data".*

Those statements are, in fact, in tune with our expectations from the current stage of development were we could only use the context factors known from the baseline data in NaPiRE. Overall, our respondents concluded along that line, which can be summarised by this exemplary statement:

> *"Detailed project characteristics need to be included [as context factors]"*

as well as by the need to further scale up from risk assessment to a holistic risk management as exemplarily reflected in this statement:

> *"Countermeasures should be included"*

In summary, the need to increase the precision and reliability of the data by adding more data points as well as context factors, and the need to extend the approach with context-sensitive recommendations in form of guidelines is in tune with our hopes and expectations expressed in Sect. 3. This strengthens our confidence in the suitability to release our approach for a large-scale evaluation which is in scope of current activities.

6 Conclusion

In this paper, we elaborated on the need for an evidence-based risk management approach that considers the particularities of RE. We argued that for such an approach, we would need a proper empirical basis on practical problems, root causes and effects in industrial RE. To lay a first foundation for such a risk management approach, we presented a cyclic research design that makes use of empirical data gathered from the NaPiRE project yielding a first approach for an evidence-based RE risk assessment. We implemented this risk assessment approach and conducted a first validation by interviewing 6 respondents from different companies.

The results of the validation have shown potential for improvement, but it also strengthens our confidence in the maturity of our solution proposal and its suitability to be transferred into practice and applied in a large-scale evaluation. In particular, the feedback from the practitioners indicate an increase of awareness for individual risk factors in RE which so far could not be provided by existing traditional risk management approaches. They also strengthened our confidence in continuing the envisioned future work. This work includes running a field study which is currently in preparation and enriching our approach with a set of recommendations associated with context-sensitive risk factors as shown in previous work on guidelines to prevent RE problems [20,25].

Acknowledgements. We are grateful to all practitioners who participated in the evaluation and who shared their experiences and insights into their environments.

References

1. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, pp. 35–46. ACM, New York (2000)
2. Damian, D., Chisan, J.: An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. IEEE Trans. Softw. Eng. **32**(7), 433–453 (2006)
3. Méndez Fernández, D., Wieringa, R.: Improving requirements engineering by artefact orientation. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (eds.) PROFES 2013. LNCS, vol. 7983, pp. 108–122. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39259-7_11
4. Méndez Fernández, D., Wagner, S.: Naming the pain in requirements engineering: a design for a global family of surveys and first results from Germany. Inf. Softw. Technol. **57**, 616–643 (2015)
5. Méndez Fernández, D., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetro, A., Conte, T., Christiansson, M.T., Greer, D., Lassenius, C., Männistö, T., Nayebi, M., Oivo, M., Penzenstadler, B., Pfahl, D., Prikladnicki, R., Ruhe, G., Schekelmann, A., Sen, S., Spinola, R., de la Vara, J., Tuzcu, A., Wieringa, R.: Naming the pain in requirements engineering: contemporary problems, causes, and effects in practice. Empir. Softw. Eng. **22**, 2298–2338 (2016)
6. Boehm, B.W.: Software risk management: principles and practices. IEEE Softw. **8**(1), 32–41 (1991)
7. DeMarco, T., Lister, T.: Waltzing with Bears: Managing Risk on Software Projects. Dorset House Publishing Co., Inc., New York (2003)
8. Pfleeger, S.: Risky business: what we have yet to learn about risk management. J. Syst. Softw. **53**(3), 265–273 (2000)
9. Boehm, B.: A spiral model of software development and enhancement. Computer **21**(5), 61–72 (1988)
10. Kontio, J.: Risk management in software development: a technology overview and the riskit method. In: Proceedings of the 21st International Conference on Software Engineering, pp. 679–680. ACM (1999)
11. Karolak, D., Karolak, N.: Software Engineering Risk Management: A Just-in-Time Approach. IEEE Computer Society Press, Los Alamitos (1995)
12. ISO 31000:2009 risk management-principles and guidelines. International Organization for Standardization, Geneva, Switzerland (2009)

13. Felderer, M., Schieferdecker, I.: A taxonomy of risk-based testing. Softw. Tools Technol. Transf. **16**(5), 559–568 (2014)
14. Kuhrmann, M., Diebold, P., Münch, J., Tell, P., Garousi, V., Felderer, M., Trektere, K., McCaffery, F., Prause, C.R., Hanser, E., Linssen, O.: Hybrid software and system development in practice: waterfall, scrum, and beyond. In: Proceedings of the International Conference on Software System Process ICSSP (2017)
15. Asnar, Y., Giorgini, P., Mylopoulos, J.: Goal-driven risk assessment in requirements engineering. Requir. Eng. **16**(2), 101–116 (2011)
16. Haisjackl, C., Felderer, M., Breu, R.: RisCal-a risk estimation tool for software engineering purposes. In: 2013 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 292–299. IEEE (2013)
17. Lawrence, B., Wiegers, K., Ebert, C.: The top risk of requirements engineering. IEEE Softw. **18**(6), 62–63 (2001)
18. Kalinowski, M., Spinola, R., Conte, T., Prikladnicki, R., Méndez Fernández, D., Wagner, S.: Towards building knowledge on causes of critical requirements engineering problems. In: Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE) (2015)
19. Méndez Fernández, D., Wagner, S., Kalinowski, M., Schekelmann, A., Tuzcu, A., Conte, T., Spinola, R., Prikladnicki, R.: Naming the pain in requirements engineering: comparing practices in Brazil and Germany. IEEE Softw. **32**(5), 16–23 (2015)
20. Kalinowski, M., Felderer, M., Conte, T., Spínola, R., Prikladnicki, R., Winkler, D., Fernández, D.M., Wagner, S.: Preventing incomplete/hidden requirements: reflections on survey data from Austria and Brazil. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2016. LNBIP, vol. 238, pp. 63–78. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27033-3_5
21. Wagner, S., Méndez Fernández, D., Felderer, M., Kalinowski, M.: Requirements engineering practice and problems in agile projects: results from an international survey. In: Proceedings of the XX Ibero-American Conference on Software Engineering (CIbSE) (2017)
22. Wieringa, R., Moralı, A.: Technical action research as a validation method in information systems design science. In: Peffers, K., Rothenberger, M., Kuechler, B. (eds.) DESRIST 2012. LNCS, vol. 7286, pp. 220–238. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29863-9_17
23. Gorschek, T., Garre, T., Larsson, S., Wohlin, C.: A model for technology transfer in practice. IEEE Softw. **23**(6), 88–95 (2006)
24. Kalinowski, M., Curty, P., Paes, A., Ferreira, A., Spinola, R., Méndez Fernández, D., Felderer, M., Wagner, S.: Supporting defect causal analysis in practice with cross-company data on causes of requirements engineering problems. In: Proceedings of the 39th International Conference on Software Engineering (2017)
25. Mafra, P., Kalinowski, M., Méndez Fernández, D.M., Felderer, M., Wagner, S.: Towards guidelines for preventing critical requirements engineering problems. In: 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (2016)

Requirement-Based Testing - Extracting Logical Test Cases from Requirement Documents

Harry M. Sneed[1,2(✉)]

[1] Technical University of Dresden, Dresden, Germany
Harry.Sneed@T-Online.de
[2] SoRing Kft., Budapest, Hungary

Abstract. Much has been written on the subject of model-based testing, i.e. taking the test cases from the system model. The prerequisite to that is that the testers have access to a design model, preferably in UML. Less has been published on the subject of requirement-based testing, i.e. taking the test cases directly from the requirement document, which is usually some structured text. Model-based testing has, to a certain extent, already been automated. Tools process the model XMI schema to extract logical test cases or test conditions as they are referred to in the ISO Standard-29119. There has yet to be any tools to extract test cases from requirement documents since this entails natural language processing and presupposes the marking up of the documents. The author began his research in this field already in 2003 while working as tester in a large scale financial services project in Vienna. There he was able to generate several thousand logical test cases and store them in the project test case database. His first internationally published paper on the subject was at the QSIC Conference in 2007. Since then he has published another five papers on the subject, alone and with others. At the heart of this approach is a text analysis tool named TestSpec which processes both English and German language requirement texts. The tool has been used in several projects to set up a logical test case database and to determine the extent of the planned system test. By counting the number of test conditions contained in the requirements text, it is possible to better estimate test effort. In all cases it proves to be helpful to testers in recognizing and defining test cases. It saves the effort involved in manually scanning through large text documents often containing several hundred pages in search of features to be tested. What used to take many days can now be done within a few minutes with the help of the tool. Not only that, but the tool also generates a test plan and a test specification in accordance with the ISO/IEEE standard. In this paper the automated approach to extracting test conditions from natural language text is explained and how it is implemented by the tool. In this praxis-oriented paper four test projects are cited which deal with very different types of requirement documents.

Keywords: Requirement documentation · Test automation
Natural language processing · Test conditions · Logical test cases
Test case types · Test case descriptions
Functional and non-functional requirements · Use cases specifications

1 Background of This Work

The author has been involved for some years now in estimating the cost of proposed systems. He receives the requirement document, referred to in German as the "Lastenheft" and is expected to measure the scope of the implementation using some requirement metric such as function-points, object-points or use case-points. To accomplish this he only has a couple of days since the average user is not willing to pay more for cost estimation. For this purpose the author has developed a text processing tool which counts the objects, the nouns used, the states defined, the actions prescribed and the use-cases. The most reliable measure is that of the use-cases. Users are more prone to define use-cases more accurately than anything else. As a minimum they will identify the use-cases and describe them in a short text. If they are diligent, they will also identify the use-case paths and their steps. Each step is an action which should be carried out. It may be that the inputs and outputs of the use-case are also part of the use-case description, making it possible to count function-points but this is seldom the case. If the user insists on counting function-points, this can only be done indirectly by transposing use-case points into function-points by means of some statistical averages on the relationship of use-cases to function-points. Object-points are counted by counting the nouns used in the use-case descriptions. If test cases are to be counted, this can only be done by identifying the use-cases and then counting the actions, states and conditions referred to in the use-case descriptions [1].

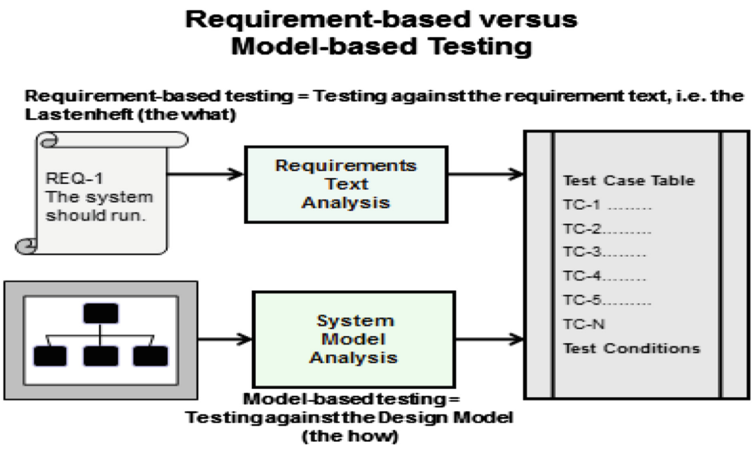
Of the 12 requirement documents that the author has been called upon to analyze in the past two years:

- one for recording all the practicing medical doctors in the country
- one for recording all the motor vehicles in the country
- one for prescribing registration cashier machines for the country
- one for prescribing a toll collection system or the country
- one for prescribing a document archiving system for the country
- one for prescribing an employee payment system for two states
- one for administering the medical insurance of all retirees in the country
- one for the marketing, production planning, distribution and post sales services of hearing aids
- one for the mobile care of disabled and elderly citizens of a large city
- one for the school administration of a large city
- one for administering the disposal of electronic garbage
- one for room management in a major university.

Nine of them contained some form of use-case description. In the other three the author had to insert dummy use-cases to be able to identify events. It is principally possible to identify potential test cases without use-cases. In this case the test cases are linked to the requirements, but there is no connection to the physical IT system. Requirements must be somehow linked to the planned system. This is where the use case comes into play [2].

2 Model-Based Versus Requirement-Based Testing

Model-based testing uses some subset of the design model to extract test cases from. It could be the activity diagrams or the sequence diagrams or a combination of both. It could also be the use-case diagrams. In industrial practice it is more likely that the users will rely on use-cases to prescribe the implementation of their proposed systems. Use-cases are the link between the requirement documentation and the system design. The system design with its diverse diagram types – sequence, diagram, activity diagram and state diagram – is seldom complete and often not done at all, but more and more use-cases are used to identify and describe the transactions performed by a planned system [3].



Requirements are simply wishes of the user. There is no way to know if and how these wishes will be fulfilled. One could attach the requirements to the object of a system but there is no guarantee that these objects will really be used. Therefore it is best to link the requirements to the real system by means of use-cases. A given requirement must be fulfilled by at least one use-case in order to be considered as satisfied. Whether it satisfied properly is another question, but at least it is assigned to be satisfied [4].

Of those 12 systems whose requirements were analyzed by the author, only three of them were later designed. For the rest a fully designed model was never created. The code was created on the basis of the requirement document, i.e. the prescribed use-cases. In most cases class diagrams were used to sketch the architecture of the system and in some activity diagrams were used to depict the flow of control, but it never got to the sequence and state diagrams. The only documents which could be used for specifying test cases were the use-case descriptions and the requirement texts of the requirements assigned to those use cases.

Thus, as we can see here, the problem with model-based testing is that there is seldom an adequate model to base the test on. There is only a requirement document and that too is more often incomplete. By analyzing the text of the requirements it is possible to recognize actions, states and conditions. For each action, a logical test case, or test condition, is generated to test if that action is really executed. For each state, a logical test case is generated which checks if that state actually occur. For each condition two logical test cases are generated – one for the positive outcome and one for the negative outcome. This leads to a large number of logical test cases ordered by requirement. But these test cases are not connected with the executable IT system. They are simply an indicator of what should be tested [5].

To connect requirement test cases to the real system, use-cases are needed. Requirements are assigned to particular use-cases to be fulfilled. If a functional requirement is not assigned to a use-case, it is assumed be out of the system. If a non-functional requirement such as a response time limitation is not assigned to a particular use-case, it should always be fulfilled, i.e., it applies to all use-cases. Behind each use-case is a set of elementary code units – procedures, functions or methods, executed in a prescribed sequence. They may be physically distributed among many different code components but at runtime they are included in the same execution path of one or more use-cases. In this way the requirements and with them their test cases are connected to the code. This is the essence of requirement-based testing. The test cases are aimed at the requirements but via the requirements also at the use-cases which fulfill those requirements [6]. The association chain is:

$$\text{Test Case} \rightarrow \text{Requirement} \rightarrow \text{Use-Case} \rightarrow \text{Code}$$

As pointed out above, test cases can also be aimed directly at Use-Cases without any reference to the requirements.

$$\text{Test Case} \rightarrow \text{Use-Case} \rightarrow \text{Code}$$

3 Generating Logical Test Cases

The focus of this paper is on identifying and generating logical test cases, or test conditions as they are referred to in the ISO-29119 standard [7]. A test condition defines a particular feature of the system. This can be an action, a state or a rule. An action would be “to place an order”. A state is “the ordered article is available”. A rule would be “the customer can only place an order if the ordered article is available”. A rule is as shown here actually a combination of a state with an action. In decision tables, the conditions are in the upper rows, the actions are in the lower rows and the columns join the two, where they cross – the intersections – are the logical test cases, i.e. the test conditions.

When given the task of defining test cases for a system test, the tester has to visually scan the requirement document to recognize the relevant actions, states and rules. Wherever the tester locates them, they are tagged as a potential logical test case. Those logical test cases are then listed out in the order of the text sections in which they are found. This could be a requirement, a use-case description, an object description or simply a text paragraph. In the end the tester produces a list of all test conditions grouped by text section and by test condition type. They can then be assigned to a particular test in which they are to be executed and to a particular tester who is responsible for testing them [8].

When automating the test condition generation, the same procedure is followed. First, the text has to be converted into a machine processable form. If it is in PDF or Word format, it has to be converted to pure text. If there are tables in the text, they have to be adjusted. If there are pictures or diagrams, they are left out. The pure text version of the requirement document is a list of variable length lines with key words to identify the line types.

The key words for marking the line types of a requirement document are not a prerequisite for identifying the test conditions [9]. Test conditions can be identified and extracted even from a formless prose text, but then they cannot be assigned to a particular requirement type. They can only be assigned to the section or paragraph in which they are found. That is why it is important to insert key words into the text before processing it. Requirements should be uniquely identified with a requirement identifier such as FREQ-01. Objects should be identified with an object identifier and data elements with a data identifier. Use-cases are to be marked with a use-case label. The attributes of a use-case, e.g. actor, trigger, precondition, post condition, path, step, etc. should also be marked as such. To be easily recognized, the key word should be placed at the beginning of the line and followed by at least one space as a delimiter.

&FUNC-REQ-04 Billing
For every customer order in which at least one item is fulfilled, an invoice is to be printed and sent to the customer. The invoices are to be prepared once a month at the end of the month. The invoice should include the items ordered, their prices, the total sum, the VAT and the amount due. All invoices should be checked by the accountant before being sent out.

&BR-01: *Unauthorized customers may not order articles.*
&BR-02: *Customers with bad credibility may not order articles.*
&BR-03: *Only articles on stock may be ordered.*
&BR-04: *An order item can only be fulfilled if the amount of articles on stock is more than the minimum amount required.*

&Actor-01: *Customer*
&Actor-02: *Accountant*
&Actor-03: *Dispatcher*
&Actor-04: *Stock Clerk*

A use case description will of course include several key words, one for every attribute as depicted below.

Attribute	Description
&UseCase	Customer_Query
Fulfill	Func-Req-01
Process	BO-01, BO-02
Input	Gui-01i
Output	Gui-01o
Implements	BR-01
&Trigger	Menu_selection
&Actor	Customer
Frequency	Continuous
PreCondition	Customer is authorized Articles are stored by category
PostCondition	Articles of the selected category are displayed by name in alphabetical order
MainPath	(1) System displays article categories (2) Customer selects a category (3) System displays all articles in that category with their prices in ascending alphabetical order
AlternatePath	None
Exceptions	System rejects customer if he is not authorized
Rules	Only authorized customers have access to the articles on stock

The key words used can be selected by the user. There is a set of some 20 predefined keywords contained in the text processing tool. However the user has the possibility of replacing these fixed four character mnemotechnical abbreviations with names of his own. For instance he can replace REQU with “Financial Requirement” or OBJT with “BusinessObject” There is a special keyword table for making this renaming. It should be filled out before starting the text analysis. Then, when the text is read, the user defined keywords will replace the literal identifiers.

SKIP = ignore lines beginning with this word
REQU = this word indicates a requirement
MASK = this word indicates a user interface
INFA = this word indicates a system interface
REPO = this word indicates a report
INPT = this word indicates a system input
OUTP = this word indicates a system output
SERV = this word indicates a web service
DATA = this word indicates a data store
ACT = this word indicates a system actor
TRIG = this word indicates a trigger
PRE = this word indicates a precondition
POST = this word indicates a post condition
PATH = this word indicates a logical path or sequence of steps
EXCP = this word indicates an exception condition
ATTR = this word indicates any user assigned text attribute

RELA = this word indicates a relation between use cases
REFS = this word indicates a reference to a requirement
RULE = this word indicates a business rule
PROC = this word indicates a business process.

When the text is read, the lines are distinguished between title lines and text lines. Title lines are recognized by having a title number such as 1.2.3 or by having a key word such as FREQ-01 or UseCase-02. The title lines mark the end of the last text section and the beginning of the next one. The sentences with a text section should end with a period or a semicolon or a question mark, i.e. the usual punctuation characters. The first action of a text parser is to build up the sentences. Once the sentences are available they are parsed token by token to recognize nouns, verbs, adjectives and conditions. Each noun is checked against a predefined noun list. Only if a sentence contains a predefined noun, i.e. a relevant object, will it be processed further. Further processing entails recognizing the verbs, adjectives, adverbs and conditional phrases. To this end the sentence structures are compared against grammar patterns. An action is recognized when a subject is combined with a verb and an object. A state is recognized if an object is linked to an adverb, such as “article is stored”, or “the bill is paid”. Rules are recognized if there is a conditional phrase in the sentence like if; when; until; as long as; in case of; should, etc. If the sentence satisfies any of these criteria, it will be cut out of the requirement text and classified as an action, a state or a rule [10].

Extracted sentences are stored in a test condition table together with their type and the section in which they appear. That section, requirement or use-case is referred to as the test target. The result is a logical test case or test condition table which can be further processed by the tester, for instance with the tool Excel. There the user may add additional attributes such as the tester name, the test vase priority and the test case status. The goal is to have a test condition for each and every feature in the proposed system. The requirement defines the feature and the logical test case tests if the requirement is fulfilled. The use-case implements the requirements with the paths and steps. The steps are described by action-type sentences from which test conditions are derived. The pre- and post-conditions are specified by state definitions from which the state-based test conditions are extracted. The exceptions are described mostly in the form of conditions. These use-case oriented test conditions are partly redundant to the requirement oriented conditions since they are describing the same functionality but from another point of view, the viewpoint of implementation. The test conditions taken from the requirements are focused on the “what”; those taken from the used-case are focused on the “how”. Together they make up a complete test of the planned system [11].

It can be that the specification of the “what” is in a separate document than the specification of the “how”, In German, the two documents are referred to as “Lastenheft” and “Pflichtenheft”. However, since the results of the automated text analysis are additive, this does not matter. The test conditions from all documents analyzed are merged together. This leads to a large number of redundant conditions which in the end have to be removed, but it is better for test conditions to be redundant than to be missing. In the German speaking world the “Lastenheft”, i.e. the required features are written by the user organization, whereas the “Pflichtenheft” is written by the supplier organization. The former document prescribes what is wanted, the second documents prescribes how it is to be achieved. This division of responsibility sounds very logical in theory, but in practice it does not always work out. As pointed out by Balzer the "how" and the "what" are very often intertwined [12]. And as Davis remarks one man's "what" is another man's "how" [13]. Besides, the "how" may not match to the "what" and this is not discovered until the test. Comparing the test conditions extracted from the two different documents is a good and inexpensive way of finding out if they are consistent.

Checking the "how" against the "what" is best done by matching their respective test conditions. If a test condition for a use-case contains the same nouns and the same verbal expressions as a test condition for a requirement, the requirement is considered to be fulfilled by that use-case. By matching their respective test conditions, use-cases can be traced to the requirements and vice versa, the requirements to the use-cases. This is also a first step to measuring requirement test coverage. To be covered a functional requirement must be assigned to a specific use-case. This is discovered by matching their test conditions [14].

Generating test conditions or logical test cases from the requirement document serves therefore at least three purposes:

1. It is the basis for defining the ultimate system test cases,
2. It enables the matching of the proposed implementation against the desired features,
3. It provides a metric for estimating the costs of system testing.

It has now been recognized that testing makes up for over 50% of the project costs [15]. The costs of testing depends to a great extend upon the number of test cases that have to be run. That member can be derived from the number of the test conditions. As a rule there are double as many physical test cases as there are logical test cases. If users know their test productivity in executed test cases per person day, they will be able to project the total test effort required from the number of logical test cases. Thus, there are three factors to be considered in estimating test costs:

- The number of logical test cases
- The relation of logical to physical test cases
- The number of physical test cases per tester day [16].

4 Transforming Logical into Physical Test Cases

That brings us to the question of how to get from test conditions to the real executable test cases. Executable test cases are no more than logical test cases, i.e. test conditions with data. Two types of data have to be added to each test condition – the input and the expected output. The data supplied as input has to be representative of the data used in production. The data defined as output has to be calculated from the input data according to the test condition. If the input is € 1000 per month and the requirement is to raise the salary by 10%, the expected output will be € 1100. Not all of the expected outputs have to be specified, but at least the critical results should be. There is no alternative to a manual calculation other than using values from previous tests or production runs. In defining a physical test case, the tester can refer to those existing output values. Otherwise, the tester has to calculate the outputs.

In defining the input data, the user can also refer to external sources or insert the values directly into the test case. If he or she selects the first approach, the input field will contain a reference to the source of the input values for this variable. It could be that the tester has prepared a sample input screen with all the values for a particular use-case. It could be that the tester refers to a table of pre-prepared values, for instance the contents of a test database. In either case the input data has to be accessible to the tester or test driver at runtime [17].

The other alternative is for the tester to build the input data into the test case description. If the data values are compact that is not a problem. They will fit into a column of the test case table. If they are long strings, the tester should place them into separate tables and refer to them by their position or by their key. Here too the data tables must be available at testing time for the tester or test driver to access.

As we can see, it is not at all trivial to combine the test conditions with test data. The means of reaching that end go well beyond the scope of this paper. The emphasis here is on detecting and documenting the test conditions, as a prerequisite to specifying the final physical test cases. This step can be automated using existing text processing technology as demonstrated in the case studies cited here. The second step of producing executable test cases with real data is not yet automatable, at least not at the current state of technology and may never be. The human tester is required to provide representative input values and to define expected result values. This remains the biggest challenge of software testing and is certainly a subject for further research [18].

5 Case Studies from Industrial Practice

To illustrate how requirement-based testing works four sample requirement documents are referred to here.

- The requirements of an online betting system
- The requirements of a mobile phone answering service
- The requirements of an airport customs office
- The requirements of a governmental website.

In the first case the test conditions were generated from a text document in which only the requirements are listed out. This was a requirement document for a betting system. In the second case the test conditions were generated from a text document in which only the use-cases were specified. This was a requirement document for handling incoming calls on a cell phone. In the third case, the test conditions were generated from a text document which contained both a list of requirements and a specification of the use-cases. It was the requirement documentation of an airport custom office. In the fourth case the test conditions were taken from a set of use cases prescribing social welfare services offered by a governmental website. The samples demonstrate that test conditions can be extracted automatically from a wide range of requirement document types.

5.1 Generating Test Conditions for a Betting System

The system in this case study is a system for reporting the results of sporting events. The users, in this case the event reporters, log into the feed system. If they are authorized and accredited to report events, they have access to the notification screens. Depending on the type of event, they can select what reporting screen they want to use and enter their data sporting events. If the notification matches to a known event, it is accepted and stored into a notification database. There it is matched with other reports on the same event. To be acknowledged as true, three different notifications from three different sources have to match. Then the data reported is stored in a final result database to be compared with the bets placed. (see the following Sample Requirements).

[RF 43] Notifications are added to a notification interface, which contains information about

The reason of the notification
The kind of the notification
The type of the notification
The date/time of the notification
If action needed: A link to the place where the action is needed.

[RF 44] The user is able "take" a notification (locked for others),
do the action and mark the notification as "finished".
If the action is finished, the user name is stored by the notification.

From a technical point of view, the event notifications were actually emails in XML format that were queued in the order in which they came in. They were then processed sequentially by a continuously running mail reception server to be validated or rejected. The output was a database of sporting event results.

This short 32 page requirement document contained:

- 95 functional requirements
- 4 non-functional requirements and
- 12 data requirements.

Besides the requirement texts it also contained 6 prototype GUI samples, 3 logical data models and a work flow diagram. The requirements were marked with a requirement identifier RF for Functional Requirement, RN for Non-Functional Requirement and RD for Data Requirement, followed by a sequence number, The GUI samples were identified by a keyword UI and the data models by the keyword DM.

The automated text analysis revealed that this document had

- 567 sentences
- 6879 words
- 116 nouns
- 279 verbs
- 100 conditional phrases.

To be tested were

- 150 states
- 106 actions and
- 100 conditions.

This added up to a total of 356 potential test conditions, i.e. logical test cases. After the duplicates were removed, there remained 287 conditions, states and actions to be tested. These were generated in the form of a text report, an excel table and an XML file where they were ordered by requirement type and test type.

When the test itself was conducted additional test conditions were added as needed. It turned out that it was not possible to foresee all the exception conditions, so in the end there were again more than 350 test conditions which were all converted over into physical test cases. Two Austrian testers were assigned to manually execute the test cases by typing in the feed messages but they turned out to be too slow and too expensive and were replaced by Slovakian testers. It had been the intention of the author to automate the test however he was pulled out of the project before he had the opportunity to complete the XML data generation tool. This he had to do later at his own expense. Users are very reluctant to finance the development of tools, even if it is to their own benefit.

The question remains as to whether it was worth it to extract the test conditions automatically from the requirements document. The same result could have been achieved manually with the aid of an excel table, but it would have required much more time and could not have been done by one person. With the use of the requirement analysis tool it was possible to generate all the test conditions within a week. Without the tool it would have required two persons at least a month.

5.2 Generating Test Conditions for a Cell Phone Answering Service

The system described here is a multi-device subscriber for handling incoming calls on a cell phone. The subscriber receives a speech or video telephone call addressed to the master device. All phones addressed by the incoming call start ringing. The user responds to the call on one phone and the other phones stop ringing. The call is registered and booked for charging.

This cell phone system was specified by 71 separate use-cases. Each use-case had the following contents:

- general description
- main track
- alternative track
- exceptions.

The general description was accompanied by a work flow diagram depicting the flow of control. The tracks were described as a sequence of numbered steps, some of which were conditional. The average size of a step was about 4 lines. Conditional steps were somewhat longer with a text for each alternative outcome. (see Sample Use Case Steps).

Sample Use Case Steps

1) If BSG_FAX, BSG_DATA or BSG_DATA64KBIT was detected, the MultiDevice logic sends depending on the type of trigger (T-CSI or VPN dialled trigger) a sinap:CUE or a sinap:CON message, writes a Call Ticket, invokes the profile synchronization and terminates afterwards.
2) If under any condition the MTC call shall be released with a final announcement, this is applicable to BSG_SPEECH, only. For BSG_VT the MTC call will be simply released.
3) If BSG_VT has been detected, but VT_PR_ACTIVE is FALSE, i.e. no parallel ringing shall be applied, the MTC call shall be set up to the master-MSISDN, only. Note that if profile:VTActivated was not set, i.e. the subscriber receives a VT call for the first time, the flag will be set (via J-330).
4) Whenever an ODSB, a Direct Dialling to a slave card or an internal call between SIM cards of a MultiDevice subscriber is detected, the MTC will be released with a final announcement. The MultiDevice logic writes a Call Ticket, invokes the profile synchronization and terminates afterwards.

The cell phone requirement specification document had more than 300 pages with an average of 4 pages per use-case. For demonstrating the definition of logical test cases, four use-cases were selected.

The difficulty in processing the document lay in the many detailed steps of each use-case and in the fact that the actions of each step were listed out as bullet points rather than as full services. In addition to that, some steps contained data definitions similar to methods in C++.

Step (1) The SEP is initialized and the data reset

Step (2) The SEP receives a smap – IDP – and triggers the MTC Base Handling service logic

- determine basic service result
- check for VPN networking (VPN – IW = FALSE for further processing)
- normalize the receiving smap – IDP, CdPA into longest format
- checks CFU status (CFU_not_active)……

Step (3) The SEP initiates a call set up towards card0 (Master MSISDW) via smap-CON with the following parameters:

- Supression of GMS-CF
- Destination Routing Address including MNP prefix
- Melody Data
- Arm EDPs for outgoing legs.

Step (4) Message will be relayed to the MSC.

The many acronyms such as here the MSC added to the difficulty of parsing this test. Any token expressed in capital letters was treated here as a noun, i.e. as a potential object. From the four use-cases selected for the study, 257 logical test case were extracted – 37 states, 131 actions and 77 conditions. Here too the test conditions were given out in three alternative formats – as text, as XML and as CSV, ordered by use-case and step. This information proved to be very useful in specifying the final physical test cases. At least the testers knew what had to be tested. Furthermore it was useful in estimating the costs of testing based on 0.3 tester days per logical test case. This added up to 86 tester days just for these four use cases. To test all 71 use cases would have cost more than 1527 person days.

The automated extraction of test conditions from the T-Mobile requirement specification was mainly used in estimating the number of test cases required and the costs of the tests. It could not be used for generating the test cases themselves. The requirement text proved to be too inaccurate. It had to be interpreted by the human testers interacting with the mobile device. The cell phone software was developed in India. The testing was to be done in Germany. Unfortunately, the author has no information on the outcome of this test. His role was restricted to estimating the costs of the test which turned out to be more than what the cell phone provider was willing to pay. As a consequence, the contact to this customer was lost.

5.3 Generating Test Conditions for an Airport Customs Office

The third case study in extracting test conditions from requirements text has to do with the custom office at an international airport. Foreign travelers are refunded the value added tax on purchases made within the country. In this requirement document the functional requirements were listed out and marked with the identifier FR-nn. Non-functional requirements were marked with the identifier NR-nn. There were altogether 93 requirements – 70 functional and 23 non-functional.

FR-08 Customer Client Desk
The CAS provides a Customs Desk Client to provide a non-automated electronic approval of TFS transactions.
NR-09 Operation in extreme weather conditions
As the pilot country will be Finland, we have to fulfill the special requirements of Finland. Customs validation in Finland happens in Nuijamaa and Vaalimaa outside the office. Custom officers (in this case GR personnel) are going to cars / busses and controlling goods outside (-20°C)[1]. At Helsinki airport goods are controlled by check in personal in the landside. CAS-Terminal and Kiosk have to be designed for those environmental circumstances.
FR-10 Refund Transactions
The CAS-Project will create xTFS Transactions, but the paper based transaction do not influence the system described here. Nevertheless the refund project (another project in the eTFS-Program) has to assure, that a transaction is refunded if the electronic transaction is approved (independent from if the paper cheque was stamped or rejected) If the eTFS transaction is rejected then the transaction is not refunded, even if the paper check is approved = stamped. This means, that the eTFS transaction is the deciding one.

This document also contained the use-case descriptions. First, the requirements were listed out and then the use-cases for implementing those requirements. At the beginning there was a process data model which defined the entities processed and their relationships to one another.

The functional requirements were expressed in one to three precise English sentences describing an action, a state or a rule. The state requirements often listed out alternative states such as payment modes – credit card, GR-card or cash. A typical functional requirement was

FR-08 Customer Client Desk
CAS should have a Customer Desk Client to provide a non-automated electronic approval of TFS transactions.

A typical non-functional requirement was

NR-09 operations in extreme weather conditions
Since the pilot country will be Finland, the system must fulfill the special requirements of that land. Customs validation in Finland often happens outside of the office; Custom officers are going to cars/busses and controlling goods outside at -20 degree temperature. At Helsinki airport goods are controlled by check-in personnel. CAS-Terminal and Kiosk have to be designed for such environmental circumstances.

The list of requirements was followed by a series of use-case descriptions. For each use-case the following attributes were defined within a prescribed template:

- short description
- purpose e.g. to refund VAT
- primary actors involved, e.g. traveler, customs officer
- goal of primary actor
- main path with detailed steps e.g. (1) entry of the request shall be done by a machine reading device; (2) the user enters a request for refund via the keyboard.
- there are different boxes of the GUI to enter it – passport-id, country, amount involved, credit card number, etc.

Typical use cases were as such:

- search transaction set
- search approval candidate transactions
- set custom status automatically
- set custom status manually

A typical use case description appeared as follows:

UC-2 Set custom status automatically

Short description	Transactions are automatically approved with an electronic stamp or not approved. In the latter case, no custom status has been changed and they are stored as approval attempt if the calling system was the kiosk.
Level	User Goal
Primary Actors	Terminal User, Kiosk
Preconditions	Actor is logged in, System knows the transaction set and has already predetermined the status. The inputs have been validated, dynamic traveler data are complete. Approval key is available for the software.
Goal Primary Actor	Get approved transactions.

.

Main Path:	**Set custom status** If the predetermined custom status of each transaction in the transaction set is approvable (=green) OR if parameter "auto-approve subset possible" = "yes" (which means that the system can approve subsets of the transaction set) then the system sets the custom status of all green transactions to "approved". Variant 1: parameter "auto-approve subset possible" = "no" and some transactions are red

In the main path an OR condition is defined indicating that two test conditions have to be generated – one for the true branch and one for the false branch.

Since this document was marked up by keywords from the beginning by the responsible analysts, there was little the tester hat to do to prepare it for automated analysis, only some minor editing to separate text sections and to mark the use cases.

The measurement of the document revealed that there were

- 1257 English sentences
- 14.355 words
- 3698 nouns
- 356 verbs
- 238 conditional phrases
- 26 user interfaces
- 17 user reports
- 8 actors
- 26 use-cases
- 28 paths
- 47 steps.

Already it was possible to see on hand of the relationship between use-cases, paths and steps that the requirement document was incomplete. The first few use-cases were described in great detail; the later use-cases were not described at all. They only had titles. This often happens in requirement analysis projects. Since the time is limited, the analyst simply stops their work when the time runs out. This should not be allowed. If there is not enough time, then the whole requirement analysis phase should be skipped over. This document was a good example of an incomplete requirement analysis. Nevertheless, from these incomplete document 296 states, 158 actions and 238 conditions could be identified for a total of 796 logical test cases. The test conditions were stored as a text, an XML file and in an Excel table, grouped by requirement and use-case. These tables were used to estimate the testing costs and a starting point for the final test case specification.

The test of the airport customs operation was carried out and eventually automated based on the logical test cases extracted from the requirements document. Here too the automatically extracted test cases were not enough to cover the test of the whole system. Another 100 non-functional test cases were later added bringing up the total number of test cases to 900. It required more than 200 tester days to run through all the test cases using the tool Selenium. The test project was completed after two months but without testing the complete functionality. The customer preferred risking errors to paying for a more through test. The project deadline had priority over the system quality.

5.4 Generating Test Conditions for Testing a Governmental Website

The fourth project concerns an E-Government web application for informing citizens of their welfare benefits in the German State of Saxony. The Web Portal allows citizens throughout the state to log in and to submit their particular life circumstances. The citizen is then given a directory of public services available to him such as child support, financial aide, free counseling, day care centers, etc. Once a certain service is selected, the system then displays which public offices are responsible for rendering that service, where they are located, how to get there and when they are open. In addition, the appropriate forms are presented for the citizen to download and fill out to apply for that public service. In Germany, it is possible for a citizen to live well on public welfare services and many do, especially in former Eastern Germany where unemployment is extremely high. Such systems are, therefore, in great demand.

As with many modern web applications, the software was not developed from scratch, but assembled from different frameworks and existing components. The existing components were taken from other states and modified to satisfy the specific requirements of the State of Saxony. The final code amounted to circa 35k Java statements running in a J2EE environment, but was never turned over to the customer, since the development contractor was also the service and platform provider. Thus, we have a typical sample of a solution being offered as a service rather than a product.

To make sure that the service satisfied the requirements, a separate contract was awarded to an Austrian testing company to test the system. The system test was a separate sub project with its own budget and its own objectives. The objectives were to test all of the requirements, both functional and non-functional, to document all of the tests made, to report all errors found to the customer and in the end, to turn over a report summarizing the test results. A maximum of four calender months and 16 person months were allocated for the test.

The baseline for the test was a set of requirement documents with a volume of over 200 pages of German prosa text, prototypes of the 24 web pages and specifications of the 11 API and WDSL interfaces [19]. The author who was sent there to prepare the test recognized immediately that he could not specify the test cases alone in the time allocated. It took at least a day to process two pages of text. That would have meant 100 days of work just specifying test cases. To shorten the process he installed the German text processor and analyzed the texts automatically. Some adjustments had to be made to the tool, but within a week he could begin generating test conditions. In the first pass through the text the nouns or potential objects were recognized. In the second pass, the text was read again together with the list of objects. By parsing the sentences, the tool was able to recognize all of the actions performed on the objects, all of the states that objects could be in and all of the conditions, i.e. business rules, governing the behavior of the objects.

Three classes of lexical expressions were extracted from the text and used to generate functional test cases:

- actions
- states and
- rules.

An action would be to check if a state service as available. A state would be if the citizen is eligible for that state service or not. A rule begins with a predicate such as if, whenever, in so far as, in case of, etc. It queries the state of an object, as in the sentence "If the applicant is an unmarried, unemployed, mother of at least one child, she is elgible for motherhood aid." Or "As long as a citizen is elgible for motherhood aide and one child is of school age, she will receive a monthly supplement for school costs".

The first condition checked a state. The second condition linked an action to a state. Actions, states and conditions, i.e. rules, need to be tested. They were the predicates of the functional test. Therefore, a test case was generated for every action on a relevant object and for every state a relevant object could be in, as well as for every rule. In the case of rules two test cases were generated – one for the positive outcome and one for the negative outcome.

It is clear that many generated test cases were redundant. So the three sets of test cases were joined to create a common subset in which each test case was unique. Since the whole process was automated it could be done in a half day and still allow another half day for the tester to weed out the insignificant test cases. The final table of test cases was then sent to the user analysts to prioritize. In this E-Government project that took another two weeks, but it was astonishing that the user actually did this ranking critical, high, medium and low priority test cases. At the end there were over 950 test cases, which had been extracted from the three word documents. This could never have been done automatically within that time.

This particular test project turned out to be a success. Another 500 physical test cases were added during the test bringing the total test cases up to 1455. The logical test cases were supplemented by real data taken from previous citizen requests. 176 of the 1455 test cases uncovered at least one error. Altogether some 457 defects were found with an effort of 392 tester days. That low cost could never have been achieved without automating the test condition generation as recommended by Peters and Parnas [20].

6 Related Work

In comparison to model based testing there is not a whole lot of related work in this field. There are relatively few researchers dealing with requirement-based testing. In Germany Mario Friske from the Frauenhofer FIRST Institute published a paper on the generation of test cases from use case descriptions already in 2004. In his first step logical test cases or test conditions were extracted from the requirements text using a text analyzer. In the second step test data was added to the test conditions to create physical test cases. In the third step test scripts were generated which could be automatically interpreted [21]. This work of Friske never got out of the research laboratory but it is interesting to note that it took place at the same time this author was developing his first text analyzer for the recognition of test cases in industrial requirement texts. Later on Friske and Pirk refined the approach and reported on it at the 35th GI-Annual Conference under the title "Werkzeuggestützte interaktive Formalisierung textueller Anwendungsfallbeschreibungen für den Systemtest" [22]. They even received a prize for their innovative work. In Austria Michael Felderer and Armin Beer have published work on the test of requirements using defect taxonomies. The defect categories are compared with the requirement specifications to identify potential test cases with a high probability of detecting defects [23].

In the USA work on the generation of test cases from requirement specification goes back to the RXVP project in the late 1970s. The General Research Corporation developed a set of testing tools referred to as the QA-Lab for testing the software of the ballistic missle defense system. Those tools included RTest for extracting test cases from the Rnet requirements [24]. Work on that subject has been going on at universities and research institutes ever since. The University of Minnesota has established a particularly outstanding reputation for its work in this field of requirement-based testing. Besides their books and articles the research team there consisting of the professors Heimdahl, Whalen and Miller offers courses via the internet [25] For a basic introduction to the subject readers are referred to the work of Magyorodi at the web site of Bloodworth Integrated Technology [26]. For more information they are referred to the work of Offutt et al. who have established criteria for generating specification-based tests [27].

7 Conclusions and Lessons Learned

The conclusion from these projects in automated test condition generation is that the extraction of logical test cases from a semi-formal requirement text can be automated and that the automated result is no worse than if the extraction is made manually. In fact, it can be much better – complete and accurate. The main advantage is that it is faster and cheaper. To manually extract the 796 test conditions from the custom refund requirements would have taken several weeks. With a tool it only took two days – one day to mark up the document and one day to run the analysis.

Therefore, if testers are going to specify test cases at all, then it is better to use a tool to select the test conditions. There is still enough to do manually in converting the test conditions to physical test cases with real data. The selection and assignment of input data values remains a significant research challenge as is the definition of the expected output values. For that real artificial intelligence is required [28].

The essence of requirement-based testing is the derivation of test cases from the requirement document. This begins with the identification and extraction of the test conditions. Every state of objects, every processing action and every logical rule is a potential test condition and should be recorded as such. Once the test conditions have been recorded, they can then be extended to include data values representative of the target application but first the test conditions have to be specified. Doing this manually is tedious and requires significant human effort. Automating it is a big step forward in making software testing more affordable.

References

1. Sneed, H.: Testing against natural language requirements. In: 7th IEEE International Conference on Software Quality, QSIC 2007, Portland, p. 380, October 2007
2. Badni, M., Badni, L., Flageol, W.: On the relationship between use cases and test suite size. ACM SIGSOFT Softw. Eng. Notes **38**(4), 1 (2013)
3. Roßner, T., Brandes, C., Götz, H., Winter, M.: Basiswissen – Modellbasierter Test. Dpunkt verlag, Heidelberg (2010)
4. Davis, A.: Software Requirements – Analysis and Specification. Prentice-Hall, Englewood Cliffs (1990)
5. Miriyala, K., Harandi, M.: Automatic derivation of formal software specifications from informal descriptions. IEEE Trans. S.E. **17**(10), 1126 (1991)
6. Poston, R.: “Action-Driven Test Case Design” in Tutorial: Automating Specification – Based Software Testing. IEEE Press, Los Alamitos (1996). p. 47
7. Daigl, M., Glunz, R.: ISO-29119 – Softwaretest Normen verstehen und anwenden. Dpunkt verlag, Heidelberg (2016)
8. Potts, C., Takahashi, K., Anton, A.: Inquiry-based requirements analysis. IEEE Softw. **12**(2), 21 (1994)
9. Bach, J.: Reframing requirements analysis. IEEE Comput. **32**(6), 113 (2000)
10. Miller, T.W.: Data and Text Mining – A Business Application Approach. Prentice-Hall, Upper Saddle River (2005)
11. Haumer, P., Pohl, K., Weidenhaupt, K.: Requirement elicitation and validation with real world scenarios. IEEE Trans. S.E. **24**(12), 1089 (1998)
12. Swartow, W., Balzer, R.: On the inevitable intertwining of specification and implementation. Comm. ACM **25**(7), 438 (1982)
13. Davis, A.: System phenotypes. IEEE Softw. **21**(4), 54 (2003)
14. Manning, C., Schütze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999)
15. Beizer, B.: Software Testing Techniques, p. 107. van Nostrand Reinhold, New York (1983)
16. Sneed, H., Seidl, R., Baumgartner, M.: Der Systemtest. Hanser Verlag, München-Wien (2008)
17. Fraser, M., Kumar, K., Vaisnavi, V.: Informal and fomal requirement specification languages – bridging the gap. IEEE Trans. S.E. **17**(5), 454 (1991)

18. Sneed, H.: Bridging the concept to implementation gap in software system testing. In: Proceedings of IEEE QSIC Conference, Oxford, G.B., p. 56, August 2008
19. Sneed, H.: Testing an eGovernment website. In: Proceedings of the WSE 2005, Budapest, p. 3. IEEE Computer Society Press, September 2005
20. Peters, R., Parnas, D.: Using test oracles generated from program documentation. IEEE Trans S.E. **28**(2), 146 (2002)
21. Friske, M.: Testfallerzeugung aus Use-Case Beschreibungen. GI Softwaretechnik Trends **24** (3), 20 (2004)
22. Friske, M., Pirk, H.: Werkzeuggestützte interactive Formalisierung textueller Anwendungsbeschreibungen für den Systemtest. GI Softwaretechnik Trends **35**(1), 27 (2015)
23. Felderer, M., Beer, A.: Eine industriell erprobte Methode für den Review und Test von Anforderungen mit Hilfe von Fehlertaxonomien. GI Softwaretechnik Trends **34**(1), 5 (2014)
24. Ramamoorthy, C.V., Ho, S.F.: Testing large software systems with automated software evaluation systems. IEEE Trans. S.E. **1**(1), 51 (1975)
25. https://www.tutorialpoint.com/software_testing_dictionary/requirements_based_testing.htm
26. Magorodi, G.: What is requirements-based testing? Bloodworth Integrated Technologies, March 2003
27. Offutt, J., Xiong, Y., Liu, S.: Criteria for generating specification-based tests. In: Proceedings of the of 5th IEEE International Conference on Engineering of Complex Computer Systems, p. 72, October 1999
28. Wilson, W., et al.: Automated quality analysis of natural language requirement specifications. In: 19th International Conference on Software Engineering, ICSE 2007, Montreal, May 2007, p. 158 (1997)

Crowdsourcing in Software Engineering

Expert Sourcing to Support the Identification of Model Elements in System Descriptions

Marta Sabou(✉), Dietmar Winkler, and Sanja Petrovic

Institute of Software Technology and Interactive Systems,
Vienna University of Technology, Vienna, Austria
{marta.sabou, dietmar.winkler,
sanja.petrovic}@tuwien.ac.at

Abstract. *Context.* Expert sourcing is a novel approach to support quality assurance: it relies on methods and tooling from crowdsourcing research to split model quality assurance tasks and parallelize task execution across several expert users. Typical quality assurance tasks focus on checking an inspection object, e.g., a model, towards a reference document, e.g., a requirements specification, that is considered to be correct. For example, given a text-based system description and a corresponding model such as an Extended Entity Relationship (EER) diagram, experts are guided towards inspecting the model based on so-called expected model elements (EMEs). EMEs are entities, attributes and relations that appear in text and are reflected by the corresponding model. In common inspection tasks, EMEs are not explicitly expressed but implicitly available via textual descriptions. Thus, a main improvement is to make EMEs explicit by using crowdsourcing mechanisms to drive model quality assurance among experts. *Objective and Method.* In this paper, we investigate the effectiveness of identifying the EMEs through expert sourcing. To that end, we perform a feasibility study in which we compare EMEs identified through expert sourcing with EMEs provided by a task owner who has a deep knowledge of the entire system specification text. *Conclusions.* Results of the data analysis show that the effectiveness of the crowdsourcing-style EME acquisition is influenced by the complexity of these EMEs: entity EMEs can be harvested with high recall and precision, but the lexical and semantic variations of attribute EMEs hamper their automatic aggregation and reaching consensus (these EMEs are harvested with high precisions but limited recall). Based on these lessons learned we propose a new task design for expert sourcing EMEs.

Keywords: Review · Models · Model quality assurance · Model elements
Empirical study · Feasibility study · Crowdsourcing · Task design

1 Introduction

During the design of software systems, a variety of models are created in the process of transforming requirements and/or specifications of the desired system into the corresponding software. These models include *Extended Entity Relationship* (EER) diagrams or UML model variants for designing databases and software system structures and behavior. The tasks of creating such models from software specifications and their subsequent verification to ensure their quality, i.e., through software model inspection [1], are cognitively intense tasks, that require significant time and effort investment from software engineers. In particular, large software models and large associated reference documents, such as requirement specifications, are challenging to inspect with limited resources in one inspection session as overly long sessions typically lead to cognitive fatigue [6]. Thus, a typical inspection session is scheduled for about two hours; large artifacts have to be scheduled in multiple inspection sessions to achieve sufficient coverage of the inspection object. Therefore, reference documents and inspection objects need to be split accordingly. Furthermore, different inspection tasks can be distributed among a group of domain experts for inspection. *Human Computation and Crowdsourcing* (HC&C) mechanisms can help splitting the workload and distributing inspection tasks among a group of experts [9, 10].

In general, HC&C techniques rely on splitting large and complex problems into multiple, small and easy tasks solvable by an average contributor in a suitable population and then coordinating the collection and aggregation of individual micro-contributions into a larger result [8]. As a result, we defined and introduced a novel *Crowdsourcing-based Software Inspection* (CSI) process, previously described in [9, 10] and illustrated in Fig. 1.

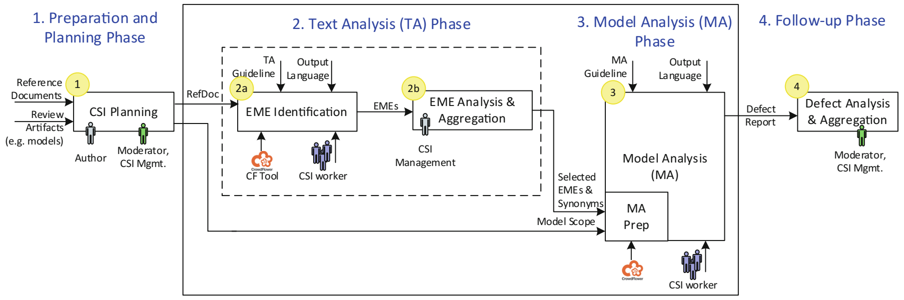


Fig. 1. Crowd-based software inspection (CSI) process [10].

The CSI process covers the *Preparation* and *Software Inspection* core steps of the traditional inspection process and consists of four fundamental phases: (1) *CSI Preparation and Planning*; (2) *Text Analysis* (TA) of reference documents to identify model building blocks, i.e., Expected Model Elements (EMEs) and aggregation of identified EMEs; (3) *Model Analysis* (MA) to identify candidate defects in the software model; and (4) *Follow-up* for defect analysis and aggregation.

During the *Preparation and Planning Phase* (Phase 1), the moderator performs inspection planning. He additionally fulfills the CSI management role. His tasks include (a) scoping of inspection artifacts, (b) preparing the crowdsourcing environment, and (c) uploading reference documents and inspection artifacts into the crowdsourcing platform, such as *Crowdflower*[1].

[1] www.crowdflower.com.

The goal of the *Text Analysis (Phase 2)* is the elicitation of key components (i.e., building blocks or co-called Expected Model Elements (EMEs)) of the model based on the reference document. EMEs include entities, attributes, and relations, that are present in system requirements specifications (i.e., reference documents) and need to be modeled in the corresponding software model, e.g., in an EER diagram. Table 1 in Sect. 4 lists typical EMEs extracted from the description of a restaurant billing system, representing the reference document (i.e., a requirements specification). Key EME classes include correct EMEs, synonyms and variations, and incorrect EMEs (see Sect. 4 for details). Note that the EME identification phase is supported by a crowdsourcing application. This phase focuses on two sub-phases: (2a) identification of candidate EMEs by a group of inspectors. We experiment with a task design where inspectors are shown one specification sentence at a time and asked to provide, based on an input syntax, the EMEs appearing in that sentence (see Fig. 4 for a task screenshot). Results are sets of candidate EMEs provided by every crowd worker in this sub-phase; (2b) EME analysis and aggregation where the moderator compiles an agreed set of EMEs based on individual results. Result is an agreed set of EMEs as input for the model analysis phase of the CSI process (Phase 3). Note that in this paper we focus on phase 2b, i.e., the EME identification and aggregation step.

In the *Model Analysis* phase *(Phase 3)*, inspectors verify the model itself (e.g., an EER diagram), or a subset thereof, while being guided by EMEs. In other words, EMEs are used as anchors for splitting the model verification task into smaller instances that can be solved in a distributed fashion according to HC&C principles. Output of this phase is a set of candidate defect lists provided by individual inspectors during their crowdsourcing task. The role of the CSI management in phase 3 is to prepare the model or a sub-model to be inspected (based on CSI planning definitions). In the *Follow-Up* Phase (Phase 4), the CSI management aggregates reported defects. Output of this final phase is a set of defects as a consensus of individual defect lists. Note that this task is completed by the CSI management.

To investigate the efficiency and effectiveness of defect detection in context of model inspection with CSI, we have conducted a large-scale empirical study involving university students as study participants in the fall of 2016. We have already reported initial findings of the CSI process approach regarding effectiveness and efficiency of defect detection when guided by high quality EMEs. High quality EMEs have been identified by the study authors [9, 10]. In context of the CSI process approach EME identification is the main output of the Text Analysis process phase. In this paper, we turn our attention to the feasibility of this text analysis task. Namely, we want to assess the feasibility of distributing the task of identifying EMEs in textual documents that describe a system specification to a number of experts as opposed to being solved by a single expert. Another key goal is to use lessons learned to improve the design of the text analysis task. For evaluation purposes, we perform a partial analysis of the CSI feasibility study reported in [9, 10] with focus on the text analysis output. We look at the EME's identified in one of 4 sessions of the feasibility study and compare these EMEs to a gold standard created by the study authors (i.e., high quality EMEs used in previous investigations). We conclude that, while entity EMEs can be identified with high precision, precision heavily deteriorates for more complex EME types where more variations are possible. This points to the need of new task designs for collecting EMEs in a reliable manner.

The remainder of this paper is structured as follows: Sect. 2 presents related work and Sect. 3 introduces to the research questions. In Sect. 4 we present the setup of the feasibility study and in Sect. 5 the experimental results and discussions. Section 6 provides a discussion of the experimental results and reflects on limitations. Section 7 summarizes lessons learned for text analysis task design. Finally, Sect. 8 concludes and identifies future work.

2 Related Work

HC&C methods have been recently used to solve a diversity of Software Engineering (SE) tasks and lead to the emerging research area of Crowdsourced Software Engineering (CSE) defined as *"the act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in an open call format"* [4, 5].

The intensified interest in the application of crowdsourcing techniques in SE can be seen as a response to the appearance of mechanized labor (micro tasking) platforms such as *Amazon Mechanical Turk*[2] (AMT) or *CrowdFlower* (See footnote 1) (CF). These platforms have popularized the Micro tasking crowdsourcing model. Micro tasking differs from more traditional models of distributed work in SE, such as collaboration and peer-production, by being more scalable, thanks to enabling parallel execution of small task units by non-necessarily expert contributors [4]. As such, Micro tasking is promising to address challenges in several cognitively complex software engineering tasks, including software inspection and model element identification. Some of the benefits of the Micro tasking model are [5]: improved coordination (e.g., of inspection team members, tasks, and results), reduction of cognitive fatigue (by removing redundant work and reusing intermediate results from previous steps), increased coverage (as some parts of large artifacts might not be covered with traditional, weakly-coordinated approaches), more diversity (support for dealing with various inspection artifacts and access to a wider variety of inspectors), and accelerated processes (by parallelization of small tasks and access to more inspectors).

Because of these benefits, crowdsourcing has been applied to address a variety of tasks pertaining to all stages of the Software Development Life Cycle (SDLC) as reported by Mao *et al.* in a recent, extensive survey [5]. Based on their classification, we derived an intuitive overview of the intensity of crowdsourcing research in each SDLC stage as shown in Fig. 2. Seventeen (17) papers report work within the *Planning and Analysis* phase on tasks such as requirements acquisition, extraction and categorization. The problems from the *Design* phase have attracted less approaches, with only four papers attempting crowdsourced user interface and architecture design. *Implementation* phase specific tasks such a coding and debugging have been investigated in 26 papers. Problems that were crowdsourced from the *Testing* phase were reported by 22 papers and included usability, performance and GUI testing. Within the *Maintenance* phase, crowdsourcing was used for software adaptation, documentation and

[2] www.mturk.com.

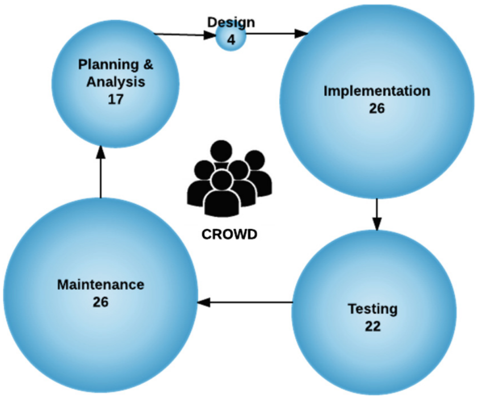


Fig. 2. Overview of number of papers classified by Mao et al. [5] as using crowdsourcing for tasks corresponding to each software development life-cycle stage.

localization among others, attracting a similar volume of approaches as the implementation phase. None of the investigated papers cover static quality assurance approaches such as software inspection, where the CSI process provides benefits for early defect detection in software models [9, 10].

However, extracting higher-level conceptual elements from text with HC&C techniques has been investigated before. For example, *Cascade* [3] is an approach to build classification hierarchies (i.e., taxonomies) from collections of independent text items (e.g., textual answers from question answering sites). An important step is to identify categories that best describe each text item. *Cascade* is designed to run on collections of independent items and it would not be suitable to extract higher-level conceptual elements (e.g., entities, relations) from ordered, logically-connected sequences of items such as sentences in a systems description document.

Approaches that can extract conceptual elements from interconnected item sets include *PhraseDetectives* [7], for co-reference resolution in text corpora, or, work on extracting categories in interconnected texts [2]. Unlike previous works, André *et al.* also focuses on tasks where domain expertise is desirable and shows that non-expert users can be supported in making sense of specialized texts [2]. This is an important aspect in the context of software inspection, which is more amenable to be solved by experts through *expert sourcing* (i.e., enlisting experts to solve tasks by using HC&C principles) rather than layman crowds.

In summary, a variety of HC&C approaches have been proposed to solve a wide range of SE tasks in all phases of the SDLC. Yet, the software model inspection task, situated within *Planning and Analysis* and *Design* SDLC phases, has not been addressed. Our work focuses on this gap and explores how HC&C micro tasking principles can be used to improve software inspection processes through expert based crowdsourcing (i.e., expert sourcing).

3 Research Questions

With focus on EME identification in context of the CSI process approach, the main research questions focus on:

RQ1: What is the overall quality of the CSI-based EME identification? To what extent can this distributed process lead to a high-quality set of EMEs? In the study context, we consider the EMEs provided by the study authors as a gold standard example of high quality EMEs.

RQ2: What are performance differences when identifying different types of EMEs? What challenges arise when distributing the identification of EMEs of increasing structural complexity? For the purpose of this paper, we define the structural complexity of an EME as the number of relations between atomic entities. For example, entities (e.g., customer) are the least complex EMEs as they refer to a single concept. Entity attributes such as *customer.name* establish a binary relation between an entity and its attribute and therefore have a higher structural complexity than entity EMEs. Relations, e.g., *(customer, orders, order)* relate three atomic entities in a triple being thus more complex than entity attribute EMEs. Relation attributes and relation cardinalities are the most structurally complex EMEs as they express more knowledge that relation EMEs.

RQ3: What are alternative expert sourcing task designs? Based on lessons learned from the analysis of the collected data, we want to identify new task design strategies that could address the limitations of the current design and lead to better results.

4 Feasibility Study Setup

The analysis reported in this paper relies on data collected in a large-scale feasibility study of the CSI-process which was described in [9, 10]. Figure 3 shows the overall experiment design with three study groups: Study group A conducts two stages of the CSI process, i.e., a text analysis and a model analysis; Study group B performs similar tasks in a different sequence; finally Study group C performs a traditional inspection process as a control group. All study groups received a tutorial at the beginning of the experiment run. Because the main goal is to investigate the effects on eliciting Expected Model Elements (EMEs) we focus on Study group A and B in this paper (Study group C does not include a task to explicitly identify and report EMEs – thus, we excluded this study group in the rest of the paper). Note that analysis results of all study groups have been described in [9, 10].

In context of this paper, the *study design* for EME elicitation consists of two main groups A and B performing the two stages of the CSI process. For group A and B, we applied a cross-over design, i.e., text analysis (60 min) followed by the model analysis (60 min) for group A and similar tasks in an inverse order for group B. This cross-over design was chosen in order to address one of the goals of the overall study, namely to assess the impact of previous domain knowledge on the defect detection performance

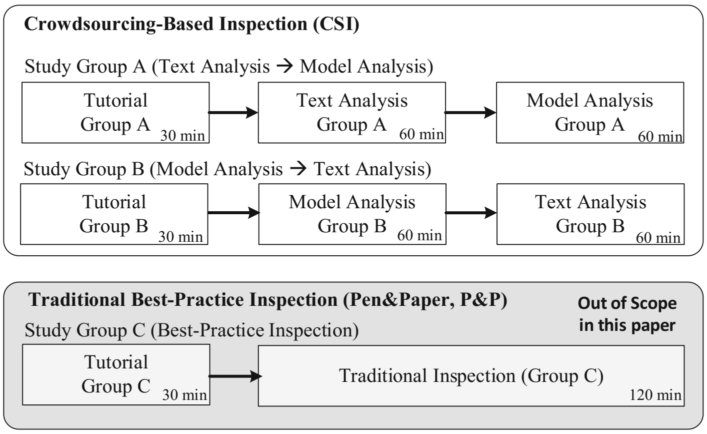

Crowdsourcing-Based Inspection (CSI)
Study Group A (Text Analysis → Model Analysis)
Tutorial Group A 30 min
Text Analysis Group A 60 min
Model Analysis Group A 60 min
Study Group B (Model Analysis → Text Analysis)
Tutorial Group B 30 min
Model Analysis Group B 60 min
Text Analysis Group B 60 min
Traditional Best-Practice Inspection (Pen&Paper, P&P)
Out of Scope in this paper
Study Group C (Best-Practice Inspection)
Tutorial Group C 30 min
Traditional Inspection (Group C) 120 min

Fig. 3. Feasibility study setup [9].

of the inspectors. In other words, we want to compare defect detection performance for inspectors that have read the specification beforehand (group A) with the performance of inspectors that detect defects only based on the EMEs and the model (group B). The time inspectors spent on each task was limited to one hour per task to (a) investigate the impact of splitting larger tasks into smaller pieces of work – a key capability of HC&C approaches, (b) enable efficient defect task execution (prior pilot runs showed that 1 h is sufficient to enable efficient EME identification and defect detection), and (c) to have a manageable experiment duration. Note that the size of the inspection artifact under investigation is suitable for a two hour inspection task, evaluated during pilot runs.

Common to all study groups is a tutorial (30 min) related to the method applied including a small practical example to get familiar with methods under investigation and related tool support. For the model analysis we used a pre-defined set of EMEs to avoid dependencies between different tasks within the experiment groups. These EMEs were provided by the experiment team, i.e., the authors. We used different experimental material for the tutorials (application domain: parking garage scenarios) and the experiment (application domain: restaurant scenarios).

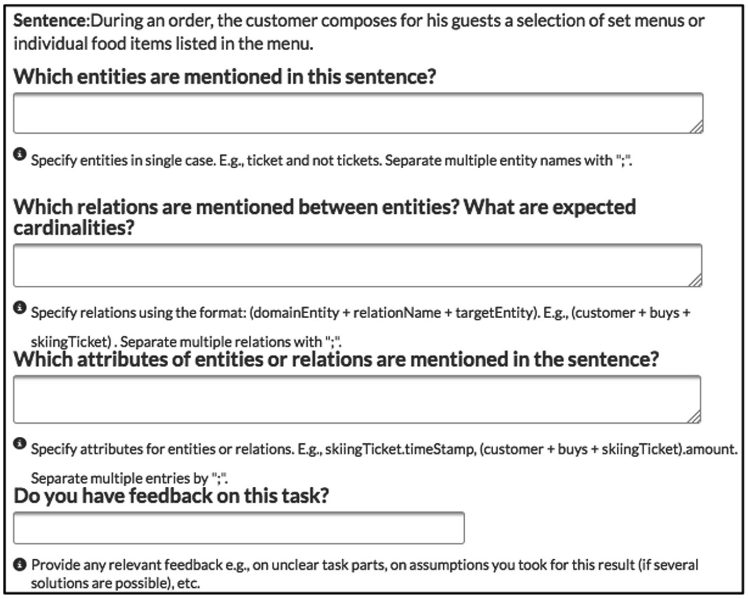
Study Population. The study was an integral part of a university course on “Software Quality Assurance” with undergraduate students at Vienna University of Technology. We captured background experience in a questionnaire before the study. Because most of the participants work at least part time in industry, we consider the participants as junior professionals [10]. We applied a class-room setting with an overall number of 75 participants (63 crowd workers and 12 inspectors) which attended 4 experiment workshops organized on different days. The group assignment was based on a random distribution of participants to study groups. Overall we had 12 study groups (four groups A, four groups B, and four croups C) during the 4 experiment workshops. Note that group C has been excluded from the analysis in context of this paper.

Study Material. We used study materials from a well-known application domain, i.e., typical scenarios and processes of a restaurant to avoid domain-specific knowledge limitations. The core study material were (a) a *textual reference document*, i.e., a system requirements specification including 3 pages in English language and consisting of 7 scenarios and (b) an *Extended Entity Relationship* (EER) diagram including 9 entities, 13 relationships, and 32 attributes. Furthermore, we used an experience questionnaire to capture background skills of the participants and feedback questionnaires after each step of experiment process. Finally, we provided guidelines that drove the experiment and the inspection process.

Gold Standard EMEs. The authors of the study identified 110 EMEs representing entities, relations, entity and relation attributes as well as relation multiplicities, which

Table 1. Example EME's from the gold standard.

EME Type 1: "Correct EMEs as those that appear in the EER model"	
Entities:	customer; order; invoice; setMenu
Relations:	(customer, orders, order) (order, orderedFoodItem, foodItem)
Entity/relation attribute	customer.name invoice.sum (customer, orders,order).date (ingredient, isProcuredBy, shoppingTour).price
Relation cardinalities	(customer(1), orders,order(0..n)) (order (0..n), orderedFoodItem, foodItem (0..n))
EME Type 2: "Synonyms and lexical variants of correct EMEs."	
Entities:	person; menu
Relations:	(foodItem, partOf, order) (order, has, foodItem)
Entity/relation attribute	Invoice.amount order.date (shoppingList, contains, ingredient).price
Relation cardinalities	(customer(1), orders, order(n))
EME Type 3: "Correct variants of modeling different from the input model"	
Entities:	dailyPlan; payment
Relations:	(foodItem, has, foodItemPrice)
Entity/relation attribute	order.status invoice.paid_date (customer, orders,order).status
Relation cardinalities	customer(1), cancels, order(0..n))
EME Type 4: "Wrong EMEs which should not be part of a correct model"	
Entities:	cook; restaurant; guest
Relations:	(order, contains, ingredient)
Entity/relation attribute	food.calorie table.seats (order, has, ingredient).amount
Relation cardinalities	(customer(1..n), orders,order(1))


Sentence: During an order, the customer composes for his guests a selection of set menus or individual food items listed in the menu.

Which entities are mentioned in this sentence?

Specify entities in single case. E.g., ticket and not tickets. Separate multiple entity names with ";".

Which relations are mentioned between entities? What are expected cardinalities?

Specify relations using the format: (domainEntity + relationName + targetEntity). E.g., (customer + buys + skiingTicket) . Separate multiple relations with ";".

Which attributes of entities or relations are mentioned in the sentence?

Specify attributes for entities or relations. E.g., skiingTicket.timeStamp, (customer + buys + skiingTicket).amount. Separate multiple entries by ";".

Do you have feedback on this task?

Provide any relevant feedback e.g., on unclear task parts, on assumptions you took for this result (if several solutions are possible), etc.

Fig. 4. CrowdFlower task interface (form-based data collection).

together constitute a set of *Gold Standard* EMEs (see some examples of EMEs from the Gold Standard in Table 1). For each type of EME, authors identified 4 categories:

1. *Correct* EMEs as those that appear in the EER model (EME Type 1).
2. *Synonyms* and *lexical variants* of correct EMEs (EME Type 2).
3. *Correct variants of modeling that are different from the input model*, i.e., the textual requirements specification (EME Type 3).
4. *Wrong* EMEs which should not be part of a correct model (e.g., superfluous) (EME Type 4).

Tooling. The *CrowdFlower* application has been used to drive the text and model analysis tasks. For the *text analysis*, the specification document was divided in sentences and participants were asked to identify EMEs in one sentence at a time. The corresponding *CrowdFlower* task contained as set of instructions and a form-based input space. The instructions explained the task in general, provided an introductory paragraph about the system that needs to be built, contained EME examples including an explanation of their specification syntax as well as some examples of sentences (from another domain) and the EMEs extracted from those sentences. The instructions are available for each task, but the instruction pane can be minimized for optimal viewing. The data collection space listed a sentence and provided forms for collecting the different EME types. Under each form, instructions were repeated about the expected naming convention and syntax.

Table 1 presents examples for identified EMEs based on the identified categories. Figure 4 illustrates and example implemented in the *Crowdflower* User Interface to identify EMEs as a crowdsourcing task. Note that we provided a so-called "Output language", i.e., a data format that unifies the data input from experiment participants.

5 Experimental Results

For the initial data analysis process we focus on the evaluation of data collected within one of the 4 experiment workshops by groups A and B participating in this workshop. Individual workshops were self-contained events and had similar numbers of participants as depicted in Table 2; the 6 participants of Group A received 14 sentences to process while the 7 participants of Group B worked on 17 sentences.

Table 2 also contains statistics about the actual numbers of EMEs collected from all participants before applying algorithms to aggregate individual contributions to each sentence as described in Sects. 5.1 and 5.2 (see corresponding Aggregation heading). Overall, when merging data from groups A and B, we collected 440 individual entity EME suggestions, which correspond to 76 syntactically unique entity EMEs. Entity and relation attributes accounted to 375 EMEs, out of which 270 were syntactically unique. Lastly, we obtained 282 relationship EMEs, with 240 syntactically unique values. An immediate observation here is that the number of recommended entities decreases as EMEs get more complex. At the same time, the percentage of unique EMEs from all EMEs is decreasing which is due to the following two factors: (1) syntactic heterogeneity increases with more complex EMEs and therefore they cannot be aggregated with syntactic string matching; (2) there is also an increased modeling heterogeneity, as participants might adapt different modeling approaches for the same information, thus hampering reaching consensus.

Table 2. Overview of experimental results in terms of identified EMEs (before/after aggregation) as well as precision and recall with respect to gold standard EMEs.

	Group A	Group B	Group A+B
Number of sentences	14	17	31
Group participants	6	7	Not relevant
Entity EMEs			
All (unique) entity EMEs	201 (42)	239 (43)	440 (76)
Entity EMEs from all sentences (unique)	25 (11)	37 (13)	62 (17)
Precision	91%	92%	88%
Recall	88%	88%	100%
Entity and relation attribute EMEs			
All (unique) attribute EMEs	137 (104)	238 (171)	375 (270)
Attribute EMEs from all sentences (unique)	17 (14)	28 (26)	45 (37)
Precision	100%	100%	100%
Recall	15%	43%	59%
All (unique) relation EMEs	114 (97)	168 (145)	282 (240)

5.1 Identification of ENTITY Model Elements

This section discusses the analysis of the entity type EMEs by both groups A and B.

Aggregation. The aggregation of EMEs suggested by individual participants took place at the level of each sentence. From the entity EMEs provided by all participants in the group for a given sentence, only the EMEs that were recommended by more than half of the participants were selected. In practical terms, we counted the frequency of each entity EME and computed a popularity score by dividing this frequency to the number of participants that rated the corresponding sentence. EMEs with popularity score higher than or equal to 0.5 were selected. To identify the EME output of a group, we created the union of sentence level entity EMEs, and selected the unique EMEs from these. Note that, in the case of entity EMEs, recommended EMEs were not pre-processed neither syntactically nor semantically. For example, we did not correct typos and did not stem the EMEs from plural to singular form. This means, that the EMEs "ingredient" and "ingredients" were treated as two syntactically different EMEs. For the final set of EMEs, we mapped these to the gold standard to determine their type (i.e., 1-correct EME, 2-synonym EME, 3-alternative EME, and 4-incorrect EME). Based on this mapping we computed the precision of the final EME set as the ratio between the EMEs of type 1–3 (i.e., correct of syntactic/semantic variations) and all identified EMEs.

For group A of the investigated workshop, a total of 25 EMEs were derived from 14 sentences based on contributions from 6 participants, corresponding to 11 unique EMEs. Based on the alignment to the gold standard we obtained precision of 91% (only 1 of 11 entities was wrongly identified). Recall was computed with respect to the 9 EMEs of type "1" (i.e., those EMEs that are part of the EER diagram). All EMEs were identified except "recipe", leading to a recall of 88%.

Group B, received more sentences therefore lead to the identification of more sentence level EMEs (37), corresponding to 13 unique EMEs in the final EME set, a precision value of 92% and recall of 88% (as in the case of group A, a single EME from the gold standard was not found, namely "invoice").

Joining the results of the two groups to obtain the EMEs for the entire reference document, we obtain a precision of 88% and a recall of 100%. Interesting observations to be drawn from here are as follows. Almost the entire set of entity EMEs was identified by each group although each group received just half of the sentence corpus (due to the cross-over design), since key entities are mentioned through the sentences. Exploiting such repetition could lead to more elegant task designs for crowdsourcing entity identification.

5.2 Identification of Entity and Relation ATTRIBUTE Model Elements

This section discusses the analysis of the entity attribute and relation attribute type EMEs collected from both groups A and B.

Aggregation. In the case of entity and relation attributes, participants contributed a large variation of syntactically diverse EMEs. Indeed, in group A, from 137 contributed EMEs (100 entity attributes, 37 relation attributes) contributed by all participants for all sentences, 104 were unique EME strings (73 entity attributes, 31 relation attributes). Therefore, given this high syntactic variation, an automatic aggregation per sentence as performed in the case of entity EMEs was not possible. Instead, we have inspected the proposed EMEs per sentence and replaced syntactic variations of the same EME to allow for aggregation. For example, for sentence Sc6S1 (the first sentence in scenario 6), participants contributed EMEs such as: *order.fullfilled*, *foodOrder.isFulfilled*, *order.fulfilledAsPlanned* which were all replaced with their semantic equivalent *order.fulfilled?*. Aggregation in terms of majority voting was performed then on the new set of EMEs. As for entity EMEs, the agreement threshold was kept at 0.5.

For Group A, a total of 17 attribute EMEs resulted from the sentence level majority voting, 14 of these were unique. All resulting EMEs were of type 1, 2, or 3, therefore the precision was 100%. However, when compared to the proposed gold standard of 39 attribute EMEs of type 1, only 6 of these were identified, leading to a recall of 15%.

From Group B, 28 attribute EMEs were collected, 26 of these being unique and all of type 1, 2, or 3. Therefore precision was 100%, while recall was 43% (17 of the identified EMEs corresponded to group 1 EMEs from the gold standard).

When merging the output EME sets from groups A and B, the EME set contains 45 EMEs out of which 37 are unique. We observe here that, unlike in the case of the entity EMEs, the overlap between the attribute EMEs derived by the two groups is quite low, as different parts of the specification point to different attributes. As a consequence, the recall of the overall EME set is 59% (23 EMEs correspond to EMEs from the gold standard).

6 Discussion and Limitations

This section focuses on the discussion of individual results towards the introduced research questions and addresses limitations and threats to validity.

6.1 Discussion

Answering our first research question (*RQ1: What is the overall quality of the CSI-based EME identification?*), we conclude that the process of distributed identification of EMEs following HC&C principles is feasible and leads to a set of EMEs that have a high overlap with a manually created gold standard set of EMEs.

Our second research question, RQ2, focused on: *What are performance differences when identifying different types of EMEs?* Based on our analysis, we conclude that the performance of verifying EMEs varies depending on the EME type:

- Entity type EMEs were identified with a high recall (100%, see Table 2) and precision (88%). We also observed that, since EME's are mentioned through the specification, a reliable set of EMEs can be extracted even just from half of the specification by either of the two study groups. This prompts to the possibility to change task design in order to more efficiently use human processing when identifying entity EMEs.

- Attribute (and relation) type EMEs were identified with a high precision (100%), but with a low recall (59%) because the syntactic and semantic variations of these EMEs are considerably higher than for entity EMEs. Because of this syntactic heterogeneity, free-text collection makes aggregation with automatic tools not feasible and opens the need for more guided task design to enable such automation in the future.
- The number of proposed EMEs decreased as the complexity of the EMEs increased. That is, while many entity EMEs were proposed (404), participants provided less entity attributes or relation attributes (375) and relations (282). Potential reasons for this phenomenon are: (1) the cognitive overload of the task is too high, as the participants are asked to do too much and therefore naturally provide less input for the more complex parts of the task; (2) writing down complex EMEs such as relations is too time-consuming and requires high effort so participant avoid it.

From the overall CSI-process perspective, the high precision of the entire EME set means that this set can act as a reliable guidance for defect detection in the follow-up model analysis task. To further improve the EME detection task and to answer RQ3 (*What are alternative expert sourcing task designs?*), we propose a novel text analysis task design in Sect. 7.

6.2 Limitations and Threats to Validity

Internal validity concerns a causal relationship between the experiment treatment and the observed results, without the influence of potential confounding factors that are not controlled or measured [11]. Domain specific issues have been addressed by selecting a well-known application domain. The experiment package was intensively reviewed by experts to avoid errors. Furthermore, we executed a set of pilot runs to ensure the feasibility of the study design [9, 10].

In terms of *external validity*, that is the generalization of the results to a larger population or to a different environment [11], we distinguish the following threats: Participants were 75 undergraduate students of computer science and business informatics at Vienna University of Technology. The study was a mandatory part of the course on “Software Quality Assurance”. Most of the participants work at least part-time in software engineering organizations. Thus, we consider them as junior professionals comparable to industrial settings. We used an experience questionnaire to capture and assess prior experiences and skills. Application domain. We used typical scenarios and requirements derived from restaurant processes. Thus, all participants are familiar with this application domain. Quality of specification documents. Admittedly, we have used a high quality specification document which has been carefully reviewed by the experiment team, i.e., the authors, during the preparation for the study. In addition we executed pilot runs of the study to ensure the quality and understandability of the study material.

7 Lessons Learned for Text Analysis Task Design

Lessons learned from this initial analysis of EMEs collected in a distributed manner prompt to several improvements at task design level, as follows (and in response to RQ3):

- *Process Improvement: Introduce a workflow of simpler tasks*. One way to overcome the complexity of the current task which hampers the acquisition of more complex EMEs, is to divide the task into a workflow of smaller tasks, each focusing on the acquisition of one EME type only.
- *Tooling: Replace free-text input tasks with more guided interfaces*. The structural complexity of the collected EMEs has an influence on two key factors. First, the higher input effort discourages participants from adding these EMEs, so less EMEs are collected. Second, the syntactic and semantic variation increases in such manner that reaching a consensus and automatic aggregation of these EMEs to allow majority voting become challenging and in some cases unfeasible. Replacing free-text input fields with more guided interfaces could overcome these limitations and (1) help EME aggregation while (2) fostering consensus building.
- *Reduce redundancy*. In the case of entity EMEs, these are frequently mentioned in several sentences of the specification and therefore lead to the redundant identification of EMEs at the cost of the participants' time and effort. Task design for entity EMEs should take this into consideration and try to reduce the redundant identification of EMEs.

Based on these envisioned improvements, we propose an updated workflow for distributed identification of EMEs, as shown in Fig. 5. The process starts with a collection of sentences from the specification which serves as input to the *Entity Identification* task to identify entity EMEs. To reduce redundancy of EME identification, we envision a task design where an initial set of EMEs are identified and aggregated from a subset of the sentences and then this set is subsequently approved and if necessary extended by participants.

Even with the current design (see Fig. 1), entity identification leads to a high quality and high coverage entity EME set. This can be used as an input to subsequent tasks for entity attribute and relation identification.

The *Entity Attribute Identification* task will consist of participants receiving a sentence and an entity EME in this sentence and being asked to enumerate attributes of that entity EME. For entities that appear in more sentences, two approaches could be considered. Firstly, if a high EME set coverage is desired, then this EME should be shown with all sentences it appears in as different sentences are likely to describe different attributes of the entity EME. Secondly, if fast execution is more desired than good coverage, the sentence to be shown could be determined according to the voting score obtained. In other words, a sentence where an entity EME was selected by many participants would be considered as most suitable for that EME and the EME would be shown only in combination with that sentence.

Each *Relation Identification* task would receive a sentence with a set of entity EMEs in this sentence and could be designed in diverse ways. In the first variant,

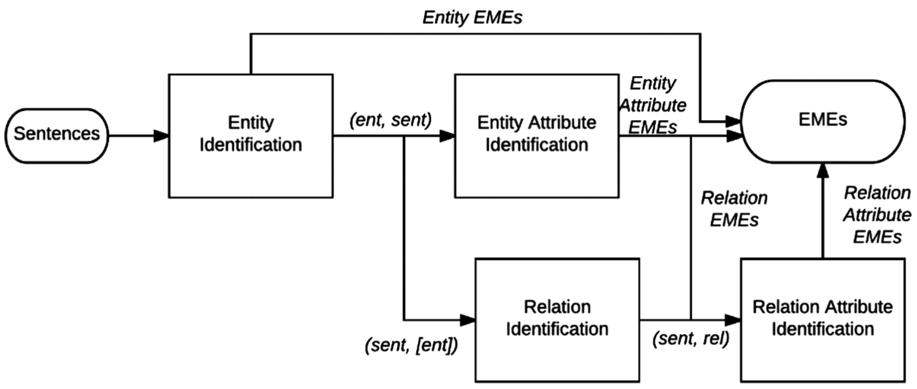


Fig. 5. Proposed workflow for distributed EME identification.

participants could be asked to provide those entity pairs between which a relation exists as free text input following a recommended syntax (with or without naming the relation). In the second variant, all combinations of entity EME pairs from a sentence could be generated and participants asked to select only those pairs between which a relation holds. This second variant has a lower effort on participants and results are easier to integrate based on selections as opposed to free-text inputs from the first variant.

Lastly, the *Relation Attribute Identification* task would benefit from the output of the Relation Identification task. For a given sentence and relation in this sentence, participants should identify potential relation attributes described in the sentence. The proposed and improved workshop process (see Fig. 5) aims at guiding participants and foster reaching consensus to a set of EMEs that can be subsequently used as input to the Model Analysis task of the CSI process. While inspecting the data, we identified in the collected responses diverse modeling choices, such as the following examples:

- *(customer(1), orders, order(0..n)).isTakeout* versus *order.isTakeout;*
- *(customer(1), has, order(0…n)).orderNumber* versus *order.number;*
- *(setMenu, contains, foodItem).price* vs. *setMenu.price* and *foodItem.price.*

With the current design, less popular modeling choices will be lost in favor of those that the majority agrees on. Yet, harvesting diverse ways to model the same specification could be beneficial for other contexts, for example when building a model rather than verifying a given model. In this context, a crowdsourcing process could identify, from a specification different models that capture a variety of views. For that purpose, new workflows should be considered that focus on preserving modeling diversity as opposed of building consensus.

8 Conclusion and Future Work

In this paper, we focus on interpreting results from the *Text Analysis* step of the *CrowdSourcing-based* Inspection (CSI) process [9, 10] which supports early defect detection of large-scale software engineering artifacts and models based on textual system specifications. The goal of the *Text Analysis* step is the distributed identification of *Expected Model Elements* (EMEs) within the system specification. EME terms represent important model elements derived from a reference document and are used as an input in the subsequent *Model Analysis* step of CSI where the defect detection in software engineering models takes place.

We analysed a subset of the data collected in a large-scale feasibility study of CSI, focusing on the comparison of the EMEs collected in a distributed fashion with a gold standard set of EMEs created by a single expert. Our key finding was that the text analysis task can lead to a highly precise set of EMEs which can serve as a suitable input for the defect detection tasks in the *Model Analysis* phase of CSI. However, while the precision of EME set is high, its recall is low for EMEs other than entities. Indeed, when harvesting more complex EMEs (e.g., entity and relation attribute), study participants provided a wide range of syntactically diverse EMEs. These leads to two difficulties. First, automatic result aggregation is challenging. Second, even after manually mapping syntactic variants of EMEs to a base form, we noticed difficulties in reaching a consensus as participants provided different EMEs based on diverse modelling views.

We concluded that these limitations are a side-effect of the current task design, which is too complex and not sufficiently guided to help consensus building. Therefore, we propose a new task design which is based on a workflow of simpler tasks, each focusing on the acquisition of one entity type. Also, we propose that these tasks are more guided: we start with the acquisition of entity EMEs which can be acquired with high recall and precision and use these to guide the following tasks. To support automatic aggregation of results, where possible, we replace free-text input with selection from a set of EMEs derived from previous steps.

Future Work. This paper reports on preliminary findings after initial manual analysis of a subset of the data. Lessons learned from this manual stage will serve to define automatic processing scripts that can be used to (semi-)automatically interpret the rest of the data and more reliably verify the results of this paper. Follow-up experiments will also test the newly proposed task design with respect to improving the effectiveness of EME identification. We will also apply the results of this work to the task of model acquisition from textual specifications. In particular, here we are interested in investigating diverse types of workflows, namely those that foster EME set diversity as opposed to fostering consensus.

Acknowledgments. We would like to thank the participants of the software quality course at Vienna University of Technology in the winter term 2016/2017 for participating in the study.

References

1. Aurum, A., Petersson, H., Wohlin, C.: State-of-the-art: software inspection after 25 years. J. Softw. Test. Verif. Reliab. **12**(3), 133–154 (2002)
2. André, P., Kittur, A., Dow, S.: Crowd synthesis: extracting categories and clusters from complex data. In: Proceedings of the Conference on Computer Supported Cooperative Work (CSCW), pp. 989–998 (2014)

3. Chilton, L.B., Little, G., Edge, D., Weld, D.S., Landay, J.A.: Cascade: crowdsourcing taxonomy creation. In: Proceedings of the Conference on Human Factors in Computing Systems (CHI), pp. 1999–2008 (2013)
4. LaToza, T.D., van der Hoek, A.: Crowdsourcing in software engineering: models. IEEE Softw. Motiv. Chall. **33**(1), 74–80 (2016)
5. Mao, K., Capra, L., Harman, M., Jia, Y.: A survey of the use of crowdsourcing in software engineering. J. Syst. Softw. **126**, 57–84 (2016)
6. NASA: Software Formal Inspection Standards, NASA-STD-8739.9, NASA (2013)
7. Poesio, M., Chamberlain, J., Kruschwitz, U., Robaldo, L., Ducceschi, L.: Phrase detectives: utilizing collective intelligence for internet-scale language resource creation. ACM Trans. Interact. Intell. Syst. **3**(1), 44p. (2013)
8. Quinn, A., Bederson, B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of Human Factors in Computing Systems (CHI), pp. 1403–1412 (2011)
9. Winkler, D., Sabou, M., Petrovic, S., Carneiro, G., Kalinowski, M., Biffl, S.: Investigating model quality assurance with a distributed and scalable review process. In: Proceedings of the 20th Ibero-American Conference on Software Engineering, Experimental Software Engineering (ESELAW) Track. Springer, Buenos Aires, Argentina (2017)
10. Winkler, D., Sabou, M., Petrovic, S., Carneiro, G., Kalinowski, M., Biffl, S.: Improving model inspection processes with crowdsourcing: findings from a controlled experiment. In: Stolfa, J., Stolfa, S., O'Connor, R.V., Messnarz, R. (eds.) EuroSPI 2017. CCIS, vol. 748, pp. 125–137. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64218-5_10
11. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wessl, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012)

Software and Systems Architecture

Are Your Requirements Covered?

Richard Mordinyi[1,2](✉)

[1] BearingPoint GmbH, Vienna, Austria
richard.mordinyi@bearingpoint.com,rmordinyi@sba-research.org
[2] SBA Research, Vienna, Austria

Abstract. The coverage of requirements is a fundamental need throughout the software life cycle. It gives project managers an indication how well the software meets expected requirements. A precondition for the process is to link requirements with project artifacts, like test cases. There are various (semi-) automated methods deriving traceable relations between requirements and test scenarios aiming to counteract time consuming and error-prone manual approaches. However, even if traceability links are correctly established coverage is calculated based on passed test scenarios without taking into account the overall code base written to realize the requirement in the first place.

In this paper the "Requirements-Testing-Coverage" (ReTeCo) approach is described that establishes links between requirements and test cases by making use of knowledge available in software tools supporting the software engineering process and are part of the software engineering tool environment. In contrast to traditional approaches ReTeCo generates traceability links indirectly by gathering and analyzing information from version control system, ticketing system and test coverage tools. Since the approach takes into account a larger information base it is able to calculate coverage reports on a fine-grained contextual level rather than on the result of high-level artifacts.

Keywords: Requirement · Testing · Test scenario · Coverage
Issue tracking system · Version control system

1 Introduction

The success of a software engineering project mostly depends on the business value it is able to produce. It is therefore essential that beside achieving quality in software artifacts team members strive for fulfilling elicited requirements. The ability to ensure that the software meets the expected requirements also depends on methods being able to report that the piece of software delivered did in fact meet all the requirements. Especially in the context of maturing software that goes into several iterations of enhancements and bug fixes, it becomes more and more daunting to ensure requirement coverage in the software [3].

Generally, requirement coverage is the number of requirements passed in proportion to the total number of requirements [27]. A requirement is considered passed if it has linked test cases and all of those test cases are passed. This implies that there has to be a traceable mapping [22] between requirements and test cases. If a link between a requirement and another project artifact, e.g. a test case, exists and this link is correct, the requirement is covered by that project artifact. If however test cases are not associated with individual requirements it could be difficult for testers to determine if requirements are adequately tested [42] leading to reported problems [10,22,26].

There are various approaches how traceability links between requirements and test cases may be identified (see Sect. 2 for details). Some of them rely on certain keywords across artifacts which are manually inserted and maintained by software developers and testers and used to establish links, while other approaches aim for full automation without human guidance by e.g., information retrieval methods. Regardless of their accuracy [14] in establishing such links, the main limitation of those methods is that links taken into consideration for requirement coverage reports reflect upon requirements and test cases as high level artifacts. While methods may access the code base during the analyzing process in order to reason about potential links they do not consider that part of the source base that actually represents and forms a specific requirement or is covered by tests [20].

However, project and test managers would like to have both information on the quality of the code base as well as on how well that quality is reflected upon the requirements which are to be fulfilled and delivered. Therefore, they need detailed, fine-grained information that allows them to reason about the progressing quality of a requirement during development phases. The quality of the code base may be measured by methods like code coverage - *the degree to which the source code of a program has been tested* [41]. However, in the context of requirements coverage we need to refine its definition as the degree to which the source code of (i.e. relevant to execute/composes) a requirement is covered by tests.

This paper describes the “Requirements-Testing-Coverage” (ReTeCo) approach that automatically establishes links between requirements and test cases by identifying the source code lines that form a requirement and by identifying the test cases which cover those source code lines. If a source code line that is covered by a test case is part of the source code line relevant for a requirement, a match has been found and a link between the requirement and the test case is created. Identification of requirement relevant source code lines is performed (a) by extracting the issue number(s) of a ticketing system which are in relation to a requirement and (b) by analyzing the log of the versioning system for changes on the code base introduced in the context of the issue. A match between a requirement and any of the test cases is given if code coverage analyzes shows that any of the identified source code lines is covered by at least one of the test cases. If a set of source code lines supports a single requirement, the number of source code lines (within that set) which are covered by test scenarios represent the percentage of coverage. Based on large open source projects we will show the feasibility of the approach and will discuss its advantages and limitations.

The remainder of this paper is structured as follows: Sect. 2 summarizes related work on requirements traceability and approaches on deriving requirements and test scenario relations. Section 3 presents research questions while Sect. 4 depicts a typical use case. Section 5 describes the ReTeCo approach. The feasibility and the initial evaluation results of the prototype implementation are illustrated in Sect. 6 and discussed in Sect. 7. Finally, Sect. 8 draws conclusions and pictures future work.

2 Related Work

Traceability is the ability to follow the changes of software artifacts created during software development [28,36] and is described by the links that map related artifacts [32]. This section summarizes related work on various methods and approaches on requirements traceability, requirements coverage by means of test scenarios, and traceability between test cases and source code.

2.1 Manually Guided Approaches

Attempts to automate the generation of traceability links concentrated on parsing the text in the code documentation to find textual relations to requirement identifiers or to the requirement descriptions [24]. Similar approaches like [34] improved accuracy by introducing specific types of comments. When text written in these comments follow some rules, the tool can trace it accurately to its requirement. The advantage of this approach is that it separates the comments written for the trace from the documentation itself. Despite their accuracy, the text parser must be very accurate and must interpret the meaning of the textual documentation to find a relation to the requirements. Even if the parser is accurate, there is no guarantee that the documentation of both the requirements and the code is up-to-date. Poor maintenance lead to wrong results and thus to higher risks undesirably increasing efforts required from developers.

2.2 Information Retrieval

In the information retrieval area, there are various models which provide practical solutions for semi-automatically recovering traceability links [4]. At first, links are generated by comparing source artifacts (e.g., requirements, use cases) with target artifacts (e.g., code, test cases) and ranking pairs of them by their similarity, which is calculated depending on the used model. A threshold defines the minimum similarity score for a link to be considered as a candidate link. These candidate links are then evaluated manually, where false positives are filtered out. The remaining correct links are called traceability links. Evaluations [12] show that this process of traceability links recovery with the aid of an information retrieval tool is significantly faster than manual methods and tracing accuracy is positively affected. Nevertheless, human analysts are still needed for final decisions regarding the validity of candidate links. Some variations of the method are depicted in the following paragraphs.

The vector space model (VSM)[4] represents all artifacts as vectors which contain the word occurrences of all vocabulary terms. The cosine of the angle between two corresponding vectors is used as the similarity measure of two artifacts. In the probabilistic model (PM)[4] the similarity score is represented as the probability that a target artifact is related to a source artifact. It was shown that similar results are achieved when preliminary morphological analysis (e.g., stemming) are performed, regardless of the used model.

The Latent Semantic Indexing (LSI)[31] extends VSM by additionally capturing semantic relations of words like synonymy (words with equivalent meaning) and polysemy (word with multiple meanings) [39]. Performance evaluations show that LSI is able to compete with the VSM and PM methods, with the advantage of not being dependent on preliminary morphological analysis. This is especially useful for languages with complex grammar (e.g., German, Italian), for which stemming is not a trivial task [29].

The problem of vocabulary mismatch occurs because IR methods assume that a consistent terminology is used, which means, that concepts are described with the same words throughout the project. However, during a projects lifecycle the used terminology usually gets inconsistent and related artifacts are not recognized anymore. This leads to declining tracing accuracy. [30] evaluates the natural language semantic methods VSM with thesaurus support (VSM-T) [25], Part-of-Speech-enabled VSM (VSM-POS) [8], LSI [13], latent Dirichlet allocation (LDA) [5], explicit semantic analysis (ESA)[18] and normalized Google distance (NGD) [9]. The authors compare the methods to the basic VSM and show that explicit semantic methods (NGD, ESA, VSM-POS, VSM-T) provide better results than latent methods (LSI, LDA) in terms of precision. On the other hand, latent methods achieve higher recall values.

In [19] a way to check requirements-to-code candidate/trace links automatically is suggested by utilizing code patterns of calling relationships. It is based on that methods or functions in the source code that implement the same requirement are related as caller and callee. As a consequence, an expected trace for a method or a function can be computed by examining neighboring caller/callee methods and their associated requirements. Invalid candidate/trace links can be detected by comparing the expected trace with the actual trace.

However, there are limitations when using IR-based traceability link recovery methods that cannot be completely solved by improvements of IR methods either. Namely, it is not possible to identify all correct trace links without manually eliminating a large number of false positives. Lowering the similarity threshold to find more correct links, will in fact lead to a strongly increasing amount of incorrect links that have to be removed manually [29].

2.3 Model-Based Techniques

Model-based requirement traceability techniques try to translate requirements to e.g., UML, XML or formal specifications. This is necessary to semi-automatically generate trace links and/or check them to consequently establish a certain degree of automation. In [1] informal requirements are restructured by means of the Systems Modelling Language (SysML) [15] into a requirement-model. These elements are manually linked with various elements of different design models, which are used for automatically deriving test cases relying on different coverage criteria and the corresponding links. In [17] trace links are generated during model transformations, which are applied to produce a design model and in further consequence discipline-specific models from a requirement model. The resulting correspondence model between source and target represents the trace links. Methods relying on UML models are for example described in [23] or [43]. The former links Use Case Diagrams and the corresponding code with the help of machine learning. Developers have to establish just about 6% trace links initially. After that the machine learning algorithm can automatically link the most of the remaining ones. The latter describes a special method for model driven development in the web engineering domain. The requirements are expressed as XML and translated by the means of XSL-transformation to an OOWS navigational model. Afterwards the links are extracted and requirement coverage is measured.

In the application of formal specifications/models requirement-, architecture- and design models are expressed for example as linear temporal logic [21], in Z-notation [40] or B-Notation [6]. The generation and/or validation of the trace links can be automated through a model checker [21], a rule based checker [40] or model based testing [6]. Further research work has been invested in techniques like annotating code, design elements or tests with traceability information [2, 14, 33], scenario-based techniques [16], graph-based techniques [7], or techniques in the context of test cases [38] relying on naming conventions, explicit fixture declarations, static test call graphs, run time traces and lexical analysis, and co-evolution logs.

3 Research Issues

The quality of software development tools and environments in supporting development has improved significantly during the last decade. While at first the effective, quality-assured support of development was one of the main concerns, nowadays tools tend to focus on better interconnecting the engineer with other information sources. They tend to interlink information [35] in any of the used tools in the project's software engineering environment as much as possible. Anyhow, effective software engineering projects cannot afford to dispense using at least a requirement modeling tool, issue tracking system, or a version control system [37] in their environment.

Since the main aim of an engineering project is to meet requirements as expected by the customer, it is essential for project managers and for engineers to know the degree of requirement coverage. Given the limitations of current requirements traceability approaches (see Sect. 2) we have formulated the following research questions:

RQ1: how to make use of links between information units provided by engineering tools for the establishment of traceability links between requirements and test cases?
RQ2: to what extent do interlinked information in engineering tools allow more fine-grained reporting on coverage?
RQ3: to what degree is an automated process for coverage calculation achievable or still require human intervention?

To answer these research issues we designed the "Requirements-Testing-Coverage" (ReTeCo) approach and implemented a prototype[1]. We then performed initial evaluations using the code base, requirements and issue sets of large and popular open source projects.

4 Use Case

Figure 1 depicts a typical software engineering process describing how requirements are "transformed" into source code. The requirements engineer is responsible for eliciting and clearly specifying the project's requirements (Fig. 1, 1). Usually, such artifacts are managed by a requirements management tool, like Polarion[2], Rational[3], or RMsis[4]. In cooperation with a release manager and usually with a member of the development team the requirements are divided into multiple working tasks (i.e. issues) (Fig. 1, 2), each having a unique identifier (i.e. issue number/id). This requires high understanding of the client specifications and the business goals alongside with high understanding of the technical abilities of the development teams. Tools for managing working tasks are for example Bugzilla[5], HP Quality Center[6], or Atlassian Jira[7]. At this point the release manager inserts a (web) link into the working task pointing to the requirement, so that any other team member is able to look up the details of the requirement in case of unclarities.

In principle, a working tasks may describe any pensum for the assignee of the task. In this context it either describes the details to be implemented by the developer (Fig. 1, 3a), or it documents the test cases to be implemented by a tester (Fig. 1, 3b).

In both cases development will be done using a version control system. Once the working task is finished the changes in the repository are committed and pushed (Fig. 1, 4a and 4b). The changes made to the code base reflect the required behaviour as described in each working task. The commit itself requires the committer to provide a commit message. Beside describing what changes were added to the code base, the committer also adds the ID of the working

[1] download available at https://github.com/mindpixel/requirementsCoverage.
[2] http://polarion.siemens.com.
[3] http://www-03.ibm.com/software/products/en/ratidoor.
[4] https://products.optimizory.com/rmsis.
[5] https://www.bugzilla.org/.
[6] https://saas.hpe.com/en-us/software/Quality-Center.
[7] https://www.atlassian.com/software/jira.

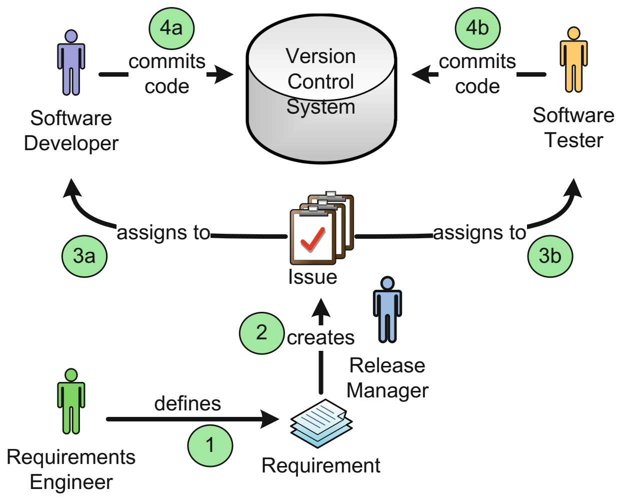


Fig. 1. Intertwining processes of various stakeholders in a software engineering environment

task to the message[8] indicating the context in which the development was done. Depending on the size of the working task or the way a developer works several commits may have been done in the context of one working task.

The release manager needs to estimate and evaluate the state of development at different phases of the project life cycle so that he/she can decide upon delivery of the software. Once all working tasks have been done, he/she asks - among other things - the following questions to ensure high-quality delivery: (a) which test scenarios check intended functionality of a requirement, (b) how many of those tests are positive, (c) how many of those tests fail, and (d) could have any test cases been overlooked?

5 Solution Approach

The following section explains the traceability link model (TLM) and the process of how to make use of it for detailed requirement coverage reports.

5.1 Traceability Link Model

The "Requirements-Testing-Coverage" (ReTeCo) approach calculates the coverage of a requirement as the degree to which the source code of (i.e. relevant to execute) a requirement is covered by tests. In order to to do it needs to rely on a traceable link between a requirement and a source code line. Figure 2 presents the relation of engineering artifacts that form the TLM for ReTeCo.

[8] an example on message structure: https://confluence.atlassian.com/fisheye/using-smart-commits-298976812.html.

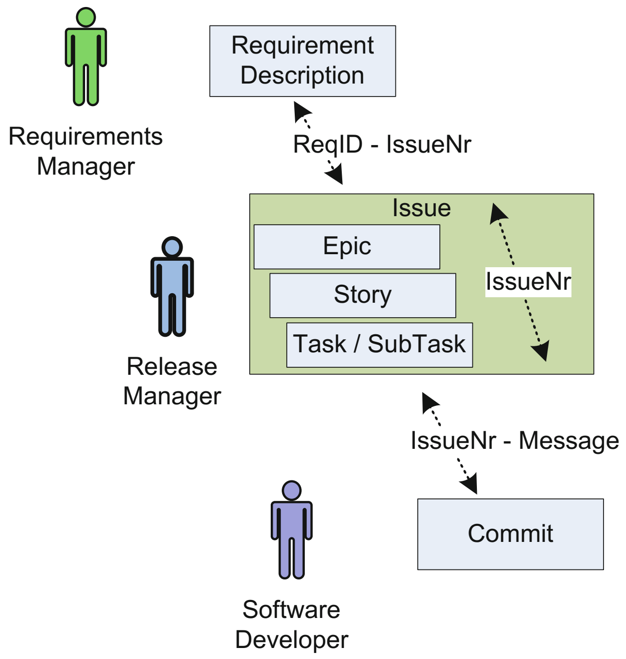


Fig. 2. Tool-supported linking of model elements define implicit traceability links

The central element of the TLM is the *Issue*. An issue is in relation to a requirement as well as to the code base. The relation between requirement and an issue is set up when the requirement is organized as a set of issues reflecting the intend of the requirement. The relation is defined within an issue containing a reference to the requirement. This means that there is a 1:n relation between the two model elements.

The relation between source code lines and an issue is set up by the developer when the developer commits the changes into the version control system introduced into the code base due to the task description in the issue. The relation is defined within the commit message by providing the issue number in that message. Since a source code line may have been altered several times, there is an n:m relation between source code and issue.

Issues may also be organized in an hierarchy. In the context of agile software development [11] it is common to distinguish between Epics, Stories, Tasks (and Sub-Tasks). An epic is a large body of work that can be broken down into a number of smaller stories. A story or user story is the smallest unit of work in an agile framework. It is a software system requirement that is expressed in a few short sentences, ideally using non-technical language. The goal of a user story is to deliver a particular value back to the customer. A task is a concrete implementation requirement for the development team. Relations between the various issues types are created (semi-) automatically whenever a sub-issue is created.

5.2 Selection of Code Coverage Tool

The ReTeCo approach relies on the analysis of coverage reports created by coverage tools. However, there are a variety of code coverage tools which differ in their capabilities and weaknesses. For the prototype implementation it is therefore necessary to set up a set of selection criteria (see also Table 1):

SC1 - Test Relation Support: In order to be able to calculate the percentage of passed and failed tests of test covered lines, it is important to have a relation model between test scenarios and source code lines. Is a coverage tool capable of providing such information? **SC2 - Export Capability:** The ReTeCo approach aimed to automatically analyze the code coverage report and extract information from the report. Is the resulting coverage report usable for further automated processing? **SC3 - License:** For an open source prototype implementation following APLv2 the compatibility of the inspected tool's license has to be checked. **SC4 - Build Process Support:** In order to increase the flexibility of the approach, it needs to be checked to what extent the tool may be embedded into a build process systems.

Table 1. Results of the criteria investigation.

Criteria & Tools	Test relation support	Export capability	License	Build process support
JaCoCo	No	HTML, XML, CSV	EPL	Maven
EclEmma	No	HTML, XML, CSV	EPL	No
Cobertura	No	HTML, XML, CSV	Free, GNU GPL	Maven
SonarQube	No	No	LGPL v3	Maven, Ant, Gradle

As shown in Table 1, there is no tool that is able to provide a relation model between tests and source code lines. For the prototype it has been decided to go with JaCoCo as it is widely used and has a solid documentation. This also means that the calculation of the relation had to be implemented explicitly.

5.3 Requirement Coverage Calculation Process

This section explains the main process step (see also Fig. 3) of the ReTeCo approach as well as implementation details of the prototype:

1. Check Consistency of the Project: In the first step the process has to ensure that project under investigation is correct and the source code can be compiled.

2. Build Source Index: In the second step the process searches all directories recursively for source code files and memorizes their locations.

3. Build Commit Index: In the third step (see also Fig. 4) the process traces each line of the source code to the Issue-ID that initially created or changed

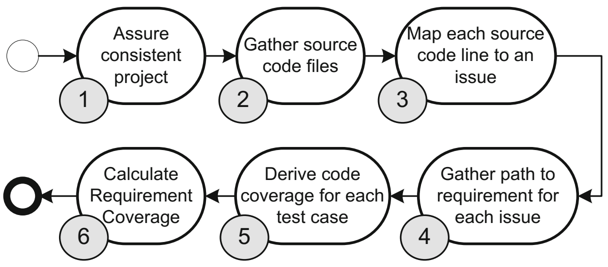

Assure consistent project
1
Gather source code files
2
Map each source code line to an issue
3
Gather path to requirement for each issue
4
Derive code coverage for each test case
5
Calculate Requirement Coverage
6

Fig. 3. Main process steps of the ReTeCo approach

that line. In the ReTeCo prototype this is done by parsing the commit (i.e. log) messages of the version management system. The prototype parses a git repository[9] of the target project by using the JGit framework[10]. It calls a series of git commands on the repository for each source code file to find out which issue is related to which source code line. At first, the prototype calls the command *git blame* for the inspected file. The result of this command contains the revision number for each line of code. Then the prototype calls the command *git log* for the each revision number. The result of running this command contains the commit message of the commit in which the line of code was modified. By parsing the commit message (e.g., using regular expression) the Issue-ID is extracted from the commit message. After repeating this process step for all lines of source code in all source files, the final outcome of this step is a set of traceable relations between source code lines and Issue-IDs.

4. Build Requirement Index: In the fourth step, each IssueID is traced back to a requirement. Under the circumstance that there is a hierarchy of issues, for each issue the corresponding parent issue is requested from the issue tracking system until the "root" issue (e.g., Epic) has been found. As explained in the previous section, the "root" issue contains the reference to the requirement it is reflecting. At this point the traceability links between source code lines and requirements have been established.

5. Perform Test Coverage: In the next step the code coverage information of the project is gathered. In this step for each line of source code it needs to be retrieved by which test case that line is covered. The outcome of this process step is a set of relations between source code lines and covering test cases. The prototype divides this task into two steps. In the first step all test cases are executed (e.g., by calling the maven[11] test goal). During the process the name of each test cases and its status (success or failure) is collected. The prototype retrieves this information by parsing the maven surefire report[12]. In the second

[9] https://git-scm.com/.
[10] https://eclipse.org/jgit/.
[11] https://maven.apache.org/.
[12] http://maven.apache.org/surefire/maven-surefire-plugin/.

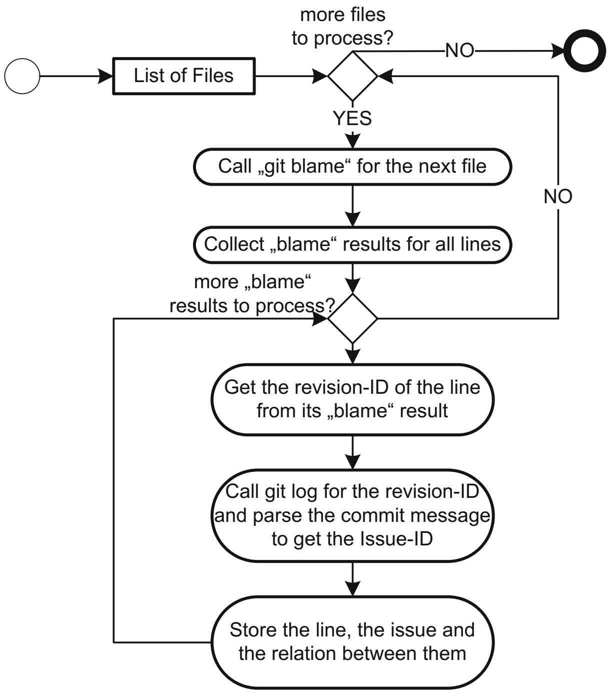

List of Files
more files to process?
NO
YES
Call „git blame“ for the next file
Collect „blame“ results for all lines
NO
more „blame“ results to process?
Get the revision-ID of the line from its „blame“ result
Call git log for the revision-ID and parse the commit message to get the Issue-ID
Store the line, the issue and the relation between them

Fig. 4. Process steps for calculating issue-code relation

step a code coverage measurement tool is run (e.g., JaCoCo[13]) to measure the test coverage information of the project. However, JaCoCo is not able to provide direct traces between a specific test case and a source code line. It only indicates if the code is traversed by any of the test cases. Therefore, code coverage reports are created for each test case separately in order to be able to relate each test case and its resulting report to a specific issue and thus to a requirement. The prototype parses the resulting JaCoCo report and extracts which lines of code are relevant to the coverage measurement.

6. Calculate Requirement Coverage: In the final step the requirement coverage report is calculated (see Figs. 5 and 6 as examples). At this point it is known (a) which source code line is related to which Issue-IDs (and therefore to which requirement) and (b) which source code line is covered by which test case. This allows the process to start calculating the coverage for each requirement. Calculation is done by analyzing and aggregating the results of each test coverage report in the context of the corresponding issues and source code lines. The final outcome of this process step contains the coverage information of the entire project - for each requirement, and for each issue of each requirement.

[13] http://www.eclemma.org/jacoco/.

6 Initial Evaluation

In the following we demonstrate the feasibility of our approach by illustrating initial evaluation results. In order to investigate and evaluate the approach, we have implemented a prototype(see footnote 1) and analyzed its performance in sense of execution time and memory consumption. We have evaluated the ReTeCo approach in the context of two open source projects ops4j paxweb[14] and Apache qpid[15]. Table 2 depicts the key characteristics of each of the projects: size of the code base, number of test cases, number of issues, and number of commits.

The evaluations were conducted on a Intel Core i7-2620M with 2,7 GHz and 8 GB RAM running on a Windows 7 environment. The prototype was implemented in java and compiled with Java 8. It uses JGit(see footnote 10) v4.4 to execute git commands, Maven Invoker[16] v2.2 in order to execute maven goals on target projects, JaCoCo(see footnote 13) v0.7.9 as the code coverage measurement tool to create the code coverage reports of the target project, and JFreeChart[17] v1.0.14 to create pie-charts showing the final requirements coverage reports.

Table 2. Characteristics of investigated open source projects

Project	# of source code lines	# of test cases	# of issues	log size
org.ops4j.pax.web	82705	63	1229	3979
org.apache.qpid.qpid-java-build	481112	1775	1684	7768

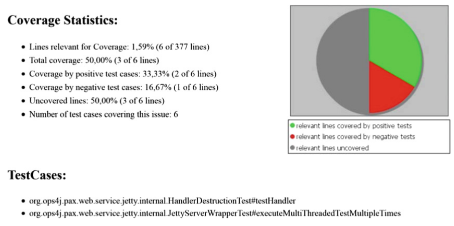
Table 2 shows that the qpid project has 5,8 times more source code lines (even half the number of commits) and 28 times more test cases than the paxweb project. Although qpid is a larger project the execution of the prototype requires only relatively little more amount of RAM, as shown in Table 3. However, the process execution time is in case of qpid significantly greater than in case of paxweb. Anyhow, the reason for the large execution time is given due to the limitation of JaCoCo. JaCoCo is not capable of directly (i.e. on the first run) reporting traces between test cases and a source code lines. Each test case had to be run separately, leading to high execution times.

Figure 5 shows collected information about the coverage of a single issue which is in relation to 6 test cases. It shows that while 377 lines of code were written or updated in the context of that issue, only 6 of them are relevant for coverage analysis. Left out of consideration are lines such as comments, javadoc, import statements, or configuration files. 50% of those lines are covered by tests - 2 lines (33,33%) positively (i.e. tests pass) and 1 line (16,67%) negatively (i.e. tests fail). The rest 3 lines (50%) are not covered by tests at all.

[14] https://github.com/ops4j/org.ops4j.pax.web.
[15] https://qpid.apache.org/.
[16] https://maven.apache.org/plugins/maven-invoker-plugin/.
[17] http://www.jfree.org/jfreechart/.

Table 3. Evaluation results of investigated open source projects

Project	Execution time	Memory consumption
org.ops4j.pax.web	1 h 15 min	177 M
org.apache.qpid.qpid-java-build	2d 09 h 34 min	207 M



Coverage Statistics:

- Lines relevant for Coverage: 1,59% (6 of 377 lines)
- Total coverage: 50,00% (3 of 6 lines)
- Coverage by positive test cases: 33,33% (2 of 6 lines)
- Coverage by negative test cases: 16,67% (1 of 6 lines)
- Uncovered lines: 50,00% (3 of 6 lines)
- Number of test cases covering this issue: 6

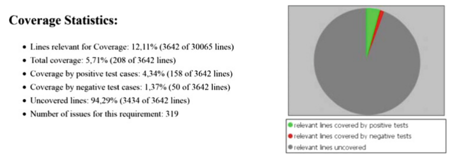
● relevant lines covered by positive tests
● relevant lines covered by negative tests
● relevant lines uncovered

TestCases:

- org.ops4j.pax.web.service.jetty.internal.HandlerDestructionTest#testHandler
- org.ops4j.pax.web.service.jetty.internal.JettyServerWrapperTest#executeMultiThreadedTestMultipleTimes

Fig. 5. Excerpt of the coverage report in the context of an issue

Figure 6 shows aggregated information related to the details of a requirement. It shows that 30065 lines of code were contributed to the requirement in 319 issues. Out of them 3642 lines of code are considered relevant. Out of the set of relevant lines 208 lines of code or 5.71% are covered by tests - 158 lines (4.34%) positively and 50 lines (1.37%) negatively. The rest of the lines 94.29% are not covered by tests.



Coverage Statistics:

- Lines relevant for Coverage: 12,11% (3642 of 30065 lines)
- Total coverage: 5,71% (208 of 3642 lines)
- Coverage by positive test cases: 4,34% (158 of 3642 lines)
- Coverage by negative test cases: 1,37% (50 of 3642 lines)
- Uncovered lines: 94,29% (3434 of 3642 lines)
- Number of issues for this requirement: 319

● relevant lines covered by positive tests
● relevant lines covered by negative tests
● relevant lines uncovered

Fig. 6. Excerpt of the coverage report in the context of a requirement

7 Discussion

The Requirements-Testing-Coverage (ReTeCo) approach aims to provide a requirements coverage report on the basis of aggregated test coverage results of each issue contributing to the composition of the requirement.

Although the approach is automatically capable of calculating the requirements coverage on a source code line bases, it depends on some preconditions. It requires from various members of a software engineering project to properly handle working task identifiers (i.e. Issue-IDs of an issue tracking system). The approach needs from the release manager to insert a reference to the requirement into the issue. In case of an issue hierarchy it needs to have ensured that the hierarchy allows unambiguous traceability from leaf issues to the root issue. Finally, it needs from developers to have the Issue-ID inserted into the commit message.

However, these tasks are not performed by a single role, but may be distributed among project members reducing the overall complexity and responsibility for each of the roles. Additionally, as described in Sect. 2 some of these tasks can be automated (e.g., by using textual comparison). Furthermore, test managers or requirement managers do not need to make any directives for developers regarding the correct nomenclature and usage of source code elements (like naming of classes, javadoc structure, or use of specific keywords) helping developers concentrate on the task described in their issue. While the correct interlinking of issues may be outsourced to the deployed issue tracking system, quality checks may be put in place which ensure: (a) root issue has a reference to a requirement (e.g., if a certain reference-type is instantiated can be queried in an issue tracking system) and (b) the commit message follows a certain regular expression pattern (e.g., check if Issue-ID is followed by message content can be executed in pre-commit git hooks).

In general uncovered requirements refer to requirements with no linked test cases. It helps project members know about test cases that should be created to cover such requirements as well. The ReTeCo approach is also capable of pointing out such requirements. However, the approach considers uncovered requirements as ones where no source code line is covered by any test case in the context of the issues composing that requirement.

The percentage value calculated by the presented approach may be misleading and has to be read with caution. There are at least two scenarios to consider:

Scenario 1: Assuming there is a group of requirements but from development point of view only with small differences between them. Usually, a developer invests a lot of time and code into realizing the first requirement while implementing the others through parameterization. This implies that there is a large number of commits and issues related to the first requirement while only little for the others. Since the changed code base is smaller for those requirements, it is therefore easier to reach higher coverage.

Scenario 2: If there is requirement under development it might be the case that the approach temporarily calculates 100% coverage. This however, may only state that the source code lines composing the requirement up to that point in time are completely covered by tests. The approach is not able to indicate when development of a requirement has finished.

The approach assumes that changes to the code base are mainly introduced in the context of an issue which was created at some point in the software engineering life cycle and represents a part of a requirement - independent of the size of the change. Consequently, the approach "assumes" that if source code lines were removed within a commit they were removed as a result of the issue's intention. For example, the removed lines may have contributed to a misbehaviour of the system (i.e. bug) and the issue's intention was to fix it.

However, there are also changes which are committed to the code base without an Issue-ID. From the authors' experience the handling of such commits is a subject of discussions among project members. An example may be the release of the software system which requires the incremental of versions numbers. Commits without a reference to an issue will be ignored by the approach.

8 Conclusion and Future Work

It is the project members responsibility to develop and deliver requirements as expected by the customer. It is also their responsibility to achieve quality in software artifacts, especially in the ones needed to fulfill the requirement. Requirements coverage indicates the number of requirements passed in proportion to the total number of requirements. A requirement is considered passed if it has linked test cases and all of those test cases are passed. This requires traceable links between requirements and test cases. While there are various approaches describing how to establish mappings (semi-) automatically, they focus on test cases as high level artifacts without taking into consideration the code base that is under test or compose a requirement.

This paper presented the "Requirements-Testing-Coverage" (ReTeCo) approach which creates traceability links between requirements and test cases through source code lines which (a) have been written in the context of an issue, (b) have been committed into a version control system, and (c) produce code coverage results. Although the approach requires manual human assistance to properly set Issue-IDs, it does not require explicit linking of requirements and test cases as it recreates traceability links implicitly through analyzing references created by various team members throughout the software development life cycle between requirements, issues, and commit message. Since the approach takes into account source code lines it is able to calculate coverage reports on a fine-grained contextual level. The paper therefore calculates requirement coverage not by passed test cases linked to that requirement but indirectly, by analyzing the lines of tested source code composing a requirement.

Initial evaluation results in the context of two open source projects showed the feasibility of the proposed approach. However, the ReTeCo approach is not able to identify relations between requirements and test cases if tests do not cover any source code line realizing a requirement.

Future work will focus on (a) combining the ReTeCo approach with approaches from related work in order to avoid manual linking of requirements with issues and issues with commits (e.g., based on textual comparison) whenever they are created or committed, (b) improving precision of the approach by considering dependencies between requirements, and (c) developing methods for handling cross-cutting aspects (e.g., quality requirements) that are related to many source code elements. From the prototype implementation point of view we intend to develop one that updates its indices incrementally on per-commit basis so that it can be embedded in a Continuous Integration and Testing life cycle.

References

1. Abbors, F., Truscan, D., Lilius, J.: Tracing requirements in a model-based testing approach. In: 2009 First International Conference on Advances in System Testing and Validation Lifecycle, pp. 123–128, September 2009
2. Ahn, S., Chong, K.: A feature-oriented requirements tracing method: a study of cost-benefit analysis. In: 2006 International Conference on Hybrid Information Technology, vol. 2, pp. 611–616, November 2006
3. Ali, N., Guhneuc, Y.G., Antoniol, G.: Trustrace: mining software repositories to improve the accuracy of requirement traceability links. IEEE Trans. Softw. Eng. **39**(5), 725–741 (2013)
4. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. **28**(10), 970–983 (2002). https://doi.org/10.1109/TSE.2002.1041053
5. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003). http://dl.acm.org/citation.cfm?id=944919.944937
6. Bouquet, F., Jaffuel, E., Legeard, B., Peureux, F., Utting, M.: Requirements traceability in automated test generation: application to smart card software validation. SIGSOFT Softw. Eng. Notes **30**(4), 1–7 (2005). https://doi.org/10.1145/1082983.1083282
7. Burgstaller, B., Egyed, A.: Understanding where requirements are implemented. In: 2010 IEEE International Conference on Software Maintenance, pp. 1–5, September 2010
8. Capobianco, G., Lucia, A.D., Oliveto, R., Panichella, A., Panichella, S.: Improving IR-based traceability recovery via noun-based indexing of software artifacts. J. Softw. Evol. Process **25**(7), 743–762 (2013). https://doi.org/10.1002/smr.1564
9. Cilibrasi, R.L., Vitanyi, P.M.B.: The Google similarity distance. IEEE Trans. Knowl. Data Eng. **19**(3), 370–383 (2007). https://doi.org/10.1109/TKDE.2007.48
10. Cleland-Huang, J., Chang, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. **29**(9), 796–810 (2003)
11. Cockburn, A.: Agile Software Development. Addison-Wesley Longman Publishing Co. Inc., Boston (2002)
12. De Lucia, A., Oliveto, R., Tortora, G.: Assessing IR-based traceability recovery tools through controlled experiments. Empir. Softw. Eng. **14**(1), 57–92 (2009). https://doi.org/10.1007/s10664-008-9090-8
13. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**(6), 391–407 (1990). https://doi.org/10.1002/(SICI)1097-4571(199009)41:6⟨391::AID-ASI1⟩3.0.CO;2-9
14. Delgado, S.: Next-generation techniques for tracking design requirements coverage in automatic test software development. In: 2006 IEEE Autotestcon, pp. 806–812, September 2006
15. Delligatti, L.: SysML Distilled: A Brief Guide to the Systems Modeling Language, 1st edn. Addison-Wesley Professional, Boston (2013)
16. Egyed, A., Grunbacher, P.: Automating requirements traceability: beyond the record replay paradigm. In: Proceedings of 17th IEEE International Conference on Automated Software Engineering, pp. 163–171 (2002)
17. Fockel, M., Holtmann, J., Meyer, J.: Semi-automatic establishment and maintenance of valid traceability in automotive development processes. In: 2012 Second International Workshop on Software Engineering for Embedded Systems (SEES), pp. 37–43, June 2012

18. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI 2007, pp. 1606–1611. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007). http://dl.acm.org/citation.cfm?id=1625275.1625535
19. Ghabi, A., Egyed, A.: Code patterns for automatically validating requirements-to-code traces. In: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 200–209, September 2012
20. Gittens, M., Romanufa, K., Godwin, D., Racicot, J.: All code coverage is not created equal: a case study in prioritized code coverage. In: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research, CASCON 2006. IBM Corp., Riverton, NJ, USA (2006). http://dx.doi.org/10.1145/1188966.1188981
21. Goknil, A., Kurtev, I., Van Den Berg, K.: Generation and validation of traces between requirements and architecture based on formal trace semantics. J. Syst. Softw. **88**, 112–137 (2014). https://doi.org/10.1016/j.jss.2013.10.006
22. Gotel, O.C.Z., Finkelstein, C.W.: An analysis of the requirements traceability problem. In: Proceedings of IEEE International Conference on Requirements Engineering, pp. 94–101, April 1994
23. Grechanik, M., McKinley, K.S., Perry, D.E.: Recovering and using use-case-diagram-to-source-code traceability links. In: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC-FSE 2007, pp. 95–104. ACM, New York (2007). http://doi.acm.org/10.1145/1287624.1287640
24. Guo, J., Monaikul, N., Cleland-Huang, J.: Trace links explained: an automated approach for generating rationales. In: 2015 IEEE 23rd International Requirements Engineering Conference (RE), pp. 202–207, August 2015
25. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans. Softw. Eng. **32**(1), 4–19 (2006). https://doi.org/10.1109/TSE.2006.3
26. Heindl, M., Biffl, S.: A case study on value-based requirements tracing. In: Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE-13, pp. 60–69. ACM, New York (2005). http://doi.acm.org/10.1145/1081706.1081717
27. Krishnamoorthi, R., Mary, S.S.A.: Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Inf. Softw. Technol. **51**(4), 799–808 (2009). http://www.sciencedirect.com/science/article/pii/S0950584908001286
28. Lago, P., Muccini, H., van Vliet, H.: A scoped approach to traceability management. J. Syst. Softw. **82**(1), 168–182 (2009). https://doi.org/10.1016/j.jss.2008.08.026
29. Lucia, A.D., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. **16**(4) 2007. http://doi.acm.org/10.1145/1276933.1276934
30. Mahmoud, A., Niu, N.: On the role of semantics in automated requirements tracing. Requir. Eng. **20**(3), 281–300 (2015). https://doi.org/10.1007/s00766-013-0199-y
31. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proceedings of the 25th International

Conference on Software Engineering, ICSE 2003, pp. 125–135. IEEE Computer Society, Washington, DC, USA (2003). http://dl.acm.org/citation.cfm?id=776816.776832

32. Maro, S., Anjorin, A., Wohlrab, R., Steghfer, J.P.: Traceability maintenance: factors and guidelines. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 414–425, September 2016
33. Ni, D.C., Martinez, J., Eccles, J., Thomas, D., Lai, P.K.M.: Process automation with enumeration and traceability tools. In: Proceedings of the IEEE International Conference on Industrial Technology 1994, pp. 361–365, December 1994
34. Ooi, S.M., Lim, R., Lim, C.C.: An integrated system for end-to-end traceability and requirements test coverage. In: 2014 IEEE 5th International Conference on Software Engineering and Service Science, pp. 45–48 (June 2014)
35. Ortu, M., Destefanis, G., Kassab, M., Marchesi, M.: Measuring and understanding the effectiveness of JIRA developers communities. In: 2015 IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics, pp. 3–10, May 2015
36. Parizi, R.M., Lee, S.P., Dabbagh, M.: Achievements and challenges in state-of-the-art software traceability between test and code artifacts. IEEE Trans. Reliab. **63**(4), 913–926 (2014)
37. Portillo-Rodrguez, J., Vizcano, A., Piattini, M., Beecham, S.: Tools used in global software engineering: a systematic mapping review. Inf. Softw. Technol. **54**(7), 663–685 (2012). http://www.sciencedirect.com/science/article/pii/S0950584912000493
38. Rompaey, B.V., Demeyer, S.: Establishing traceability links between unit test cases and units under test. In: 2009 13th European Conference on Software Maintenance and Reengineering, pp. 209–218, March 2009
39. Rosario, B.: Latent semantic indexing: an overview (2000). http://www.cse.msu.edu/cse960/Papers/LSI/LSI.pdf
40. Sengupta, S., Kanjilal, A., Bhattacharya, S.: Requirement traceability in software development process: an empirical approach. In: 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping, pp. 105–111, June 2008
41. Stanbridge, C.: Retrospective requirement analysis using code coverage of GUI driven system tests. In: 2010 18th IEEE International Requirements Engineering Conference, pp. 411–412, September 2010
42. Tahat, L.H., Vaysburg, B., Korel, B., Bader, A.J.: Requirement-based automated black-box test generation. In: 25th Annual International Computer Software and Applications Conference, COMPSAC 2001, pp. 489–495 (2001)
43. Valderas, P., Pelechano, V.: Introducing requirements traceability support in model-driven development of web applications. Inf. Softw. Technol. **51**(4), 749–768 (2009). https://doi.org/10.1016/j.infsof.2008.09.008

High Quality at Short Time-to-Market: Challenges Towards This Goal and Guidelines for the Realization

Frank Elberzhager(✉) and Matthias Naab

Fraunhofer Institute for Experimental Software Engineering IESE,
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{frank.elberzhager,matthias.naab}@iese.fraunhofer.de

Abstract. High quality and short time-to-market are business goals that are relevant for almost every company since decades. However, the duration between new releases heavily decreased, and the level of quality that customers expect increased drastically during the last years. Achieving both business goals imply investments that have to be considered. In this article, we sketch 22 best practices that help companies to strive towards a shorter time-to-market while providing high quality software products. We share furthermore experiences from a practical environment where a selected set of guidelines was applied. Especially DevOps including an automated deployment pipeline was an essential step towards high quality at short time-to-market.

Keywords: Quality · Time-to-market · Guidelines · Experiences

1 Introduction

Shorter times for delivering new software products and updated versions are more and more common. Higher speed can provide substantial business value, but only if adequate quality can be delivered. Some years ago, it was quite normal to deliver a new release once or twice a year, or in even longer intervals. Today, updates are often released many times a week, sometimes even every few seconds. Internet companies are clearly the leaders of fast releases (e.g., Amazon made changes to production on average every 11.6 s in 2011, and many Google services are released several times a week (for more details, see [1, 2]), but also more traditional companies like car manufacturers delivering safety-relevant products are following suit (e.g., Volvo cars [3] or Tesla).

With short time-to-market, two views can be distinguished:

1. Delivering the first release within a short time between project start and first release
2. Delivering frequent subsequent updated releases within short times between start of increment and release

We will mainly concentrate on the second aspect. This is far more challenging, and requires sustainable processes, roles, a tool landscape, and a certain culture among other things.

D. Winkler et al. (Eds.): SWQD 2018, LNBIP 302, pp. 121–132, 2018.
https://doi.org/10.1007/978-3-319-71440-0_7

One key reason for short time-to-market is earlier value, which can be refined into four concrete benefits:

1. Customers can use new features earlier – the company can earn money earlier
2. A company is on the market faster than competitors – the company can earn more money
3. A company gets faster feedback – the company can further improve the software product or implement new feature requests from customers
4. A company provides small updates instead of large ones – the company can reduce the risk of providing the wrong software.

The motivation for companies to achieve a shorter time-to-market is clearly business-driven (as it aims at providing business value), it is no end-in-itself. The objectives have thereby to be clearly derived. Jan Bosch stated "no efficiency improvement will outperform cycle time reduction" [4].

On the other hand, short time-to-market should usually not cannibalize the quality of the software system, at least not to a large extent. High quality is important to convince customers of the value of the software product. Throwing a new product or release onto the market without proper quality assurance, documentation, and further related tasks can speedup time-to-market, but it is not sustainable at all. Though there might be situations where it is reasonable to concentrate solely on speed, and not on quality at all (or only to a minor extent), this is not the usual case and we exclude it from our scope.

But how does the need for speed impact the company's software quality requirements? And why does development time and operation quality requirements need strong improvement? From our perspective, these are key aspects when high quality at short time-to-market should be implemented in a sustainable way. Guidelines that we present in this article outline the areas where investment is needed to make high quality happen at short time-to-market. Finally, we will discuss applicable factors for significantly reducing release times. Therefore, we concentrate on three main questions in this paper:

1. What exactly does high quality mean?
2. Where and how to invest for high quality at short time-to-market?
3. What is the applicability of high quality at short time-to-market?

We answer the first question in Sect. 2 together with our procedure how we elaborated this question. Section 3 presents our results and gives several best practices and guidelines to strive towards high quality at a short time-to-market. Section 4 gives some practical hints that help companies to implement certain guidelines, and provides some experiences from a concrete industrial context. Finally, Sect. 5 concludes the article and provides some follow up questions for researchers and practitioners.

2 Procedure and Concepts

2.1 Procedure

We started our conceptual work by creating a general view on the topic. For this, we scanned several articles and publications that already exist about this topic (e.g., [7–17]). From our analysis, a motivation that, why, and where investments need to be done, was often missing. Usually, the goal of high quality and short time-to-market is considered, and we also found recommendations for improvements (e.g., best practices). However, in this article, we distinguish stronger between different qualities, and derived concrete investment areas to achieve a certain value, respectively benefits. Based on this general picture which is presented in Sect. 2.2, we identified next areas which are influenced when a company wants to achieve high quality at short time-to-market. We identified four concrete ones, which are partly based on former project experiences with software developing companies. In order to derive concrete guidelines, we performed a two hour workshop with seven experts in areas such as architecture, process improvement, and implementation. All the detailed feedback was consolidated and resulted in six concrete guideline topics which are presented in Sect. 3. We applied some of these guidelines with respect to a practical context and gained further practical experience which is shown in Sect. 4.

2.2 Conceptual View

Optimizing either delivery time or quality is challenging in itself. High quality at short time-to-market is even more challenging, and not every company benefits equally from high-frequency releases, or would have to change too many things so that the investment would not be worth the benefit. So a company needs to start asking the following questions, describing the goals:

- What are the key reasons in the company's market and which products should be released with short time-to-market?
- What is a reasonable time-to-market that the company wants to achieve?
- What is the level of quality that should not be affected?

These business-related questions have to be roughly answered first, followed by technical and organizational questions to realize the identified goals and to check the feasibility of achieving the goals. Many new and more fine-grained questions emerge when striving towards a shorter time-to-market, e.g.

- What does an efficient release process look like?
- What software-architecture is needed?
- What is the necessary infrastructure to be provided?
- What is a reasonable team structure?
- What tools do support fast delivery?

Figure 1 illustrates the transition towards high quality at short time-to-market. One key reason why software is built is the functionality it realizes. However, only if this functionality comes with an adequate quality in the software, it is actually useful. The

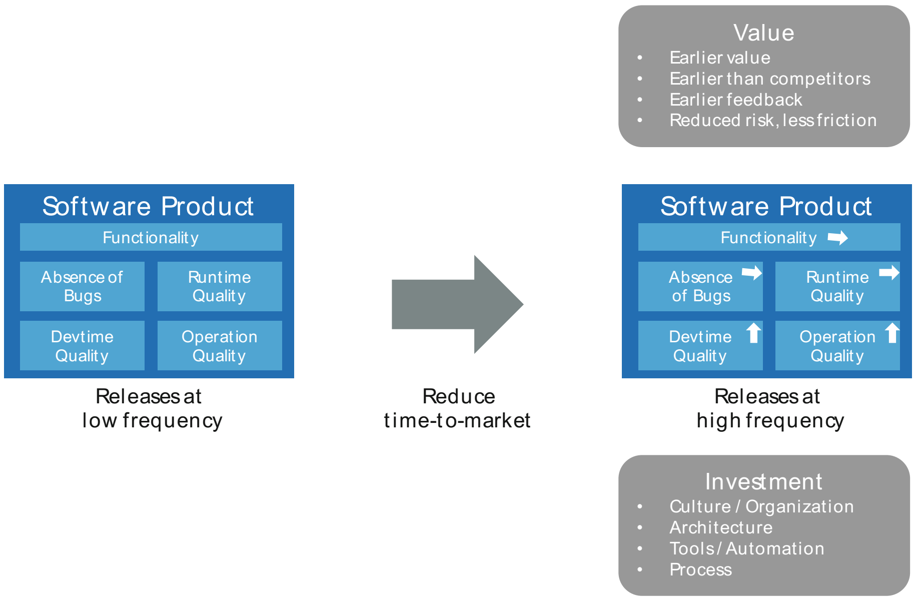


Fig. 1. Implications on quality when shifting towards reduced time-to-market

concept of quality needs thereby to be refined in order to understand what is expected to be existing by nature, and which parts need further investment. Thus, we distinguish four categories of quality characteristics of a software product:

- Absence of bugs
- Runtime quality (e.g., performance, security)
- Development time (short: Devtime) quality (e.g., maintainability, testability, extensibility, flexibility)
- Operation quality (e.g., updateability, recoverability).

The focus in development organizations is often on functionality, absence of bugs, and runtime qualities. This is quite natural, because these quality characteristics are directly visible to the customer, while devtime quality and operation quality rather serve the development organization (in particular if it also operates the software). This is also backed by the ISTQB Worldwide Testing Practice Report from 2015 reporting that the focus of quality assurance is on runtime qualities, such as performance, usability, or security [5].

High quality also means absence of bugs and runtime quality. However, in order to be able to deliver this quality characteristics with high speed over time, it becomes inevitable to also increase the devtime quality and the operation quality.

- Devtime quality is necessary to allow making additions and changes quickly (maintainability, extensibility, flexibility, etc.) and testing the changes with a high level of confidence within a reasonable period of time (which necessarily implies a large degree of automated testing).

- Operation quality is necessary to enable reliable and fast releases with a high degree of automation and to quickly react to potential failures of the newly released software.

That is, the goal is to deliver absence of bugs (or at least a minimization) and runtime quality to the customer, and the means for realizing this is to invest in devtime quality and operation quality. As functionality is available earlier, a slightly reduced amount of functionality might even be acceptable. Due to the ability to correct problems much faster than before, a bit more tolerance for bugs might exist. Figure 1 depicts this relationship. It is very important that all the people in a development organization have a clear picture of these different characteristics of quality and how they will change when they strive for more speed.

High quality at short time-to-market does not come for free. It is clearly a business value and a competitive advantage, and thus it is obvious that investments are needed. We already pointed out now two areas of quality that need strong improvement for high quality at short time-to-market: Devtime quality attributes and operation quality attributes.

Next, we broaden the view on areas of investment. For this, we have identified four high-level areas, which need investments to make high quality at short time-to-market happen:

- Culture/Organization: The people in the development organization have to be fully aware that releasing fast is of value and that everyone has to contribute. Changes to the organizational structure might be necessary to empower people to develop and release fast.
- Architecture: The architecture of the software has to strongly support fast releases, that is in particular the realization of development time and operation quality attributes (e.g. updateability, recoverability, testability) while maintaining the runtime quality attributes (e.g. performance, user experience, security, availability, …) at the same time.
- Tools/Automation: Automation is key for fast releases with high quality: Only if the quality assurance, build, and release is highly automated and reliable the releases can be done with high confidence in a high frequency.
- Processes: Processes have to focus on the full end-to-end coverage from ideation of new features to putting them into production with low friction and delay.

For each of these four high-level areas, concrete guidelines can be derived. We gathered a list of more than 40 best practices. For example, in the area of architecture, we discussed programming rules, concrete qualities such as testability that need to be coped by the concrete architecture, or encapsulation principles. Further examples for the area ‘culture’ are keeping the customer in mind or high communication skills. However, besides many obvious ideas, we consolidated all feedback into six main guideline topics and summarized concrete hints (see next Section).

3 Guidelines and Implementation

The four areas (i.e., culture, architecture, tools, and processes) which need investment are connected and do overlap. In the following, we provide more concrete topics and guidelines, which mostly touch multiple of these areas (for example, take continuous delivery: For continuous delivery, the software architecture has to cope with fast integrations and deployments, and a full process needs to be defined with a high degree of automation). Some hints how to select them and where to start is given in Sect. 3.2.

3.1 Guidelines

We derived six topics that reflect different aspects during software development. The concrete guidelines touch, as mentioned before, often not only one area, but several. Topic 1 focuses on the concrete customer and the business value, i.e. something the company always should have in mind with all the methods and procedures they apply: It is never for the software product itself, but always to serve the customer and to earn money. Customer satisfaction is not negligible, as a bad satisfaction threatens the company's success. The second topic concentrates on bringing the software product and also the company to the next level, i.e., to continuously reflect the business and all related aspect of the development. The third topic focuses more on technical issues during the release process and how to improve it. This is a major topic to really become fast. The fourth topic concentrates on means to provide high quality. Topic five then sets the focus on the whole company and reflects what is important on this higher level to support high quality at short time-to-market. Finally, topic six considers data in order to understand the current situation, but also to measure success and to identify further improvement ideas.

We do not claim that this is a complete list, but it touches main aspects that should be considered when a company wants to strive towards high quality at short time-to-market. Overall, we present 22 concrete guidelines next.

- Topic 1: Customer and business value
  - Focus on building the right software system that serves your customers and provides the highest value for your business.
  - Build as little as necessary; building less takes less time.
  - Consistently prioritize in requirements engineering.
  - Incorporate early feedback and data to enable continuous adjustment.
- Topic 2: Innovation and differentiation
  - Do not reinvent the wheel; do not invest your time into things that are already there.
  - Focus on the things that make your product and your business unique.
  - Use cloud-based technologies where possible.
  - Continuously renew: What is innovation today may be a common feature tomorrow.
- Topic 3: Release capability
  - Introduce continuous delivery, integration, and deployment for (partially) automated, and thus faster, releases.

  - Adopt DevOps practices; they aim at assuring smooth interplay between development and operation throughout the release step.
  - Design for updateability: Provide the ability to run different software product versions; use effective API version management; migrate data; etc.
  - Modularize software to enable independent releases of features and software parts: Microservices are an architectural style that proposes many decisions supporting decoupled development and releases.
  - Make teams responsible for the development, release, and operation of their modules and create the ability to release independently (respect Conway's law concerning the organizational structure).
- Topic 4: High quality investments
  - Do quality assurance early to avoid going in the wrong direction (for example, prototyping, architecture evaluation, etc.).
  - Try out concepts and features early with a strong customer focus.
  - Design for high testability and, in particular, for a high degree of automated testing.
  - Design for robustness: Provide the ability to keep failures local and to recover quickly in the case of failures.
  - Establish a culture of making even good things better over time.
- Topic 5: Overall agile development organization
  - Establish a culture of speed and fast decisions.
  - Follow agile development principles and be responsive.
- Topic 6: Utilization of data to improve business and software
  - Perform A/B tests, for example, to get early feedback about alternative features and about the quality by gradually delivering the software to the customers.
  - Ask your users for (anonymous) feedback, respectively collect data from log files.

3.2 Implementation Hints

We believe that high quality at short time-to-market will become more relevant for many industries and companies, as there is a strong ability to create business value. High quality at short time-to-market is nothing that can be bought out of the box. Rather, it is a deliberate business decision that comes with many consequences, changes and investments. To make it happen, the company developing the software has to strongly adapt and invest into the areas culture/organization, architecture, tools/automation, and processes.

The concrete manifestation differs from company to company. That starts with different releases cycles and ends at detailed technologies that are used. Such concrete environmental factors have to be considered during the definition of a concrete migration and improvement strategy.

The following aspects have an impact on the applicability and the concrete definitions of high quality at short time-to-market:

- Adherence to quality standards and certifications: Whether high release cycles are possible can be determined by regulations concerning quality assurance and certification.
- Customer expectations and ability to create value: Only if regular and fast updates are perceived to be accepted by customers and can result in a business value the investments into high quality at short time-to-market are justified.
- Status of existing software: The age and overall quality of the software (in particular the devtime quality and the operation quality) have an impact whether it is advisable to invest into high quality at short time-to-market for the current software (or rather do it for a successor)
- Control over execution environment: Companies offering software-as-a-service (SaaS) have the advantage that they have good control over the execution environment while companies offering physical goods like cars might have to update software in millions of hardware instances. However, even these things might change with strongly improving network capabilities allowing to move more and more functions to a cloud-based environment.

When a company wants to move towards high quality at short time-to-market, the following questions should initiate that journey:

- What does “high quality” mean in this context, how is “short” in time-to-market interpreted?
- What is the respective improvement in value that is expected to gain?
- What does this direction mean for the organization, the processes, the architecture, and the tools?
- Which topics are most reasonable to start with?

It should be kept in mind that it is about the balance between high quality and fast delivery, and both have to be considered to find the right balance. This means, it is about acceptable quality and it is about acceptable release frequency. The guidelines of this article can point out aspects to reason about, to prioritize, and to start introducing. They are deliberately not presented as a migration roadmap because the transition to high quality at short time-to-market will be individual and thus requires an individual migration roadmap.

4 Experiences with Selected Guidelines from the Fujitsu EST Environment

We accompanied Fujitsu EST for about one year in an improvement initiative [6]. The improvement vision of this initiative was driven by several objectives, among others the goal of higher quality and reduced release cycles (**culture of making good things better, culture of speed and fast decisions**). These goals were set at the beginning of the joined project. Concrete improvement steps could then be derived step-by-step, and a migration plan was defined based on an analysis of the current development processes, team structure, and the tool landscape.

The general focus of the joined project was to investigate the benefits of DevOps in the given context, and an implementation of selected DevOps practices. The **introduction of DevOps** was our main mission, which is reflected by one of the central aspects of the guideline list. DevOps, which wants to bring development and operations together, aims at sharing a higher joined responsibility of development and operations staff, and wants to break down barriers between these different parts and the corresponding mindsets. While DevOps in itself provides several concrete practices, it supports shorter time-to-markets by, for example, lower friction between development and operations, or much **higher automation** supported by an automated deployment pipeline. Both were major outcomes of this project, and Fig. 2 provides an overview of the deployment pipeline that was defined and implemented in the given context.

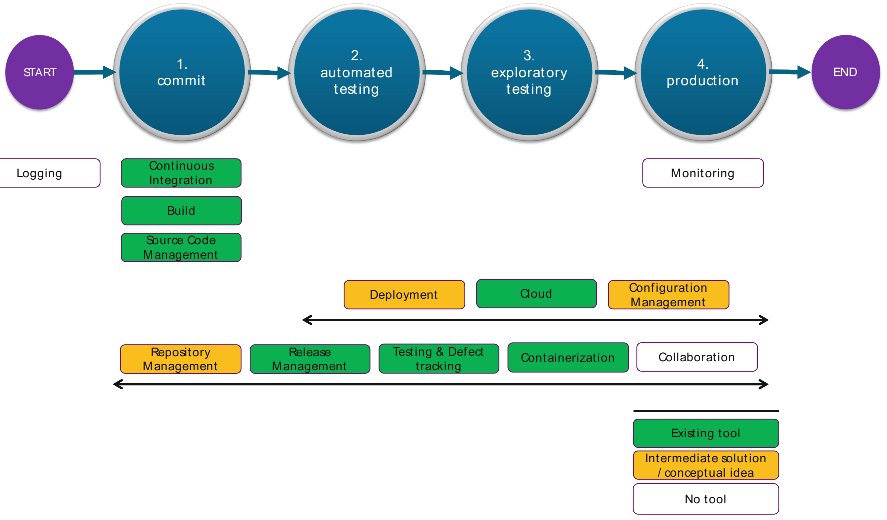


Fig. 2. Deployment pipeline and tool categories in the Fujitsu EST environment to support high quality at short time-to-market

The pipeline consists of four steps, and was – except the exploratory testing – fully automated in the end:

1. Commit: This stage starts as soon as a developer commits new or changed code to the source version control system. It invokes the **continuous integration** server, which builds the application and generates artifacts. The artifacts are stored and reused in the subsequent stages. If the build fails, the developer is notified and the commit does not progress.
2. Automated testing: The artifacts generated in the previous stage are deployed to a testing environment. Automatic testing is done for all parts of the software. The environment is provisioned and configured automatically. If the tests fail, the artifacts do not progress any further.

3. Exploratory testing: Once the automatic testing stage has been passed, the artifacts are deployed to an exploratory testing environment for manual testing. The environment is again created automatically. Once the tests are finished, the artifacts can go to the next stage; otherwise, they stop here.
4. Production: After the exploratory testing stage, the artifacts are deployed to the final production environment and users can use the application directly. The production environment is created automatically and monitored continuously. If something goes wrong, the system rolls back to a previous release. All the deployments are done following a deployment strategy. Users ideally do not experience any downtime and the transition from one release to another is smooth.

Figure 2 also presents an overview of tool categories that we considered in the pipeline. There exist numerous tools for every step, and it again depends on the concrete environment which ones fit best. Seven tool categories were already covered in the Fujitsu EST environment. For the commit stage, Jenkins was used as the continuous integration tool, Maven and Grunt for the build process, and GIT for source code management. **Services of the Google Cloud** were applied for steps 2-4. For release management, Jenkins was considered, and for containerization, Docker was the choice. Google Cloud provides the Docker management itself. For repository management, Docker Registry and Artifactory were used. Finally, for testing, quality assurance, and issue reporting purposes, jUnit, Karma, sonarQube, and Bugzilla were applied.

Collaboration was out of scope due to the small size of the team and the little current need. Also, logging was not further considered as this is done during coding, i.e., before committing and thus before the main parts of the pipeline. It was decided to focus on monitoring only after implementation of the first complete pipeline. Deployment was supported by shell scripts.

The main focus in the Fujitsu EST context to complete the pipeline thus was on configuration and repository management. For both tool categories, existing tools were collected and classified according to the requirements of the context. More than 40 configuration management tools exist, of which Puppet, Chef, Ansible, and Salt are the most popular ones considering the available success stories. The main observations were that Chef does not have a real-time configuration option, which was needed by the software product. While Chef and Puppet are mature tools compared to Ansible and Salt, they are very much developer-oriented, making it difficult for operators to handle them. On the other hand, Ansible and Salt are tools that are more oriented towards system administration and allow to write modules in any language. Thus, the decision was to try out Ansible. A similar procedure of evaluating and selecting tools based on the requirements of the context was followed for the artifact repository. In this case, Artifactory was the choice.

Besides the introduction of DevOps, high automation with several tools, and a concentration on quality aspects, we discussed some further best practices, but neglected also some. For example, we did not change the software architecture to a microservice architecture. The concepts and implications were discussed, but as the team was small and the software product rather simple, the investment was not seen as worthwhile at the current point. This might change over time, but it shows that every decision must be taken based on well-conceived reasons.

In the end and after the deployment pipeline had been set up and implemented and further improvements as sketched had been introduced, Fujitsu EST wanted to measure the effects. Fujitsu EST added six quantitative metrics to an existing dashboard. It was shown that fast deployments with high quality are now possible. About 10 min are required from a commit to the completion of automated testing. Deployment to the production stage takes about 20 min, depending on the time used for exploratory testing and the upload to the production environment in the Cloud.

As Fujitsu EST shifted more towards a service provider, and had to care about operations as well, they wanted to understand how the users experience their services. Thus, it was discussed how **feedback** could be gathered from the customer. Simple mechanisms, such as providing an email address where customers could give feedback, or a feedback bar on the webpage, were implemented.

To conclude our experience, we could improve the time-to-market with selecting a set of 6–8 of our improvement suggestions (see bold ones), and provide high quality. Many of the guidelines from our list were further refined to make them more operational. It took some investment and also learning [6], but the results in the given context were worthwhile the effort. The whole journey was supported by a migration plan to define the concrete steps needed, and to monitor the success.

5 Summary and Outlook

In this paper, we argued that for today's companies it is often needed to shift more towards a shorter timer-to-market, while still caring about high quality. There is no unique way to achieve these business goals, and for every company, the concrete instantiation means different things. In order to provide a certain starting point for such a journey, we provided a list of six improvement topics with overall 22 guidelines. Such a list is far away from being complete, but provides initial ideas in which direction a company can move, respectively what improvements might be reasonable to address. The list is mainly based on experience from software engineering experts and it is our goal to extend such a list in the future, and to provide further guidelines for the concrete application of selected guidelines. We shared the experience for a selected set of guidelines how we implemented them in the concrete Fujitsu EST environment.

There are also open issues for research from our point of view:

- What concrete guidelines will result in an as ideal as possible solution?
- What is the best order of such guidelines to be implemented, and how can a migration plan be derived based on context factors from a concrete company?
- How can the benefit be measured?

We believe that the trend towards high quality at short time-to-market will continue and that companies will have to cope with this challenge. Thereby, fast releases does not necessarily mean several times a day, but can mean to shift from monthly releases to weekly ones, for example. Research and practice therefore have to find solutions to address such needs, and guidelines as presented in this article can be an initial step – for selecting concrete improvements, but also for identifying future research directions.

References

1. Kim, G., Behr, K., Spafford, G.: The Phoenix Project – A Novel About IT, DevOps, and Helping Your Business Win. It Revolution Press (2014)
2. http://www.thoughtworks.com/de/insights/blog/case-continuous-delivery. Accessed June 2017
3. Keynote at ICSA (2017). https://youtu.be/VP1AhGGCFeI. Accessed: June 2017
4. Bosch, J.: Speed, Data, and Ecosystems: Excelling in a Software-Driven World. CRC Press (2016)
5. ISTQB Worldwide Software Testing Practices Report. http://www.istqb.org/references/surveys/istqb-worldwide-software-testing-practices-report.html. Accessed June 2017
6. Elberzhager, F., Arif, T., Naab, M., Süß, I., Koban, S.: From agile development to devops: going towards faster releases at high quality – experiences from an industrial context. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2017. LNBIP, vol. 269, pp. 33–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49421-0_3
7. Erich, F., Amrit, C., Daneva, M.: Cooperation between software development and operations: a literature review. In: ESEM (2014)
8. http://dev2ops.org/2010/02/what-is-devops/. Accessed June 2017
9. Katua, P.: DevOps from the ground up, thoughtworks. In: GOTO Conference (2014)
10. Buytaert, K.: DevOps, DevOps, DevOps. In: Froscon (2011)
11. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspectives. Addison-Wesley Professional (2015)
12. CA Technologies: DevOps: the worst-kept secret to winning in the application economy (2014)
13. Puppet labs & IT Revolution Press: 2013 State of DevOps report (2013)
14. Wickett, J.: The DevOps way of delivering results in the enterprise, Mentor Embedded (2012)
15. Kim, G.: Top 11 things you need to know about DevOps. IT Revolution Press
16. Debois, P.: Modeling DevOps, Yow Conference (2013)
17. Sharma, S., Coyne, B.: DevOps for Dummies, 2nd IBM Limited Edition (2015)

Prioritizing Corrective Maintenance Activities for Android Applications: An Industrial Case Study on Android Crash Reports

Valentina Lenarduzzi[1], Alexandru Cristian Stan[1], Davide Taibi[1,2(✉)], Gustavs Venters[1], and Markus Windegger[3]

[1] Free University of Bolzano-Bozen, 39100 Bolzano-Bozen, Italy
{vlenarduzzi, astan, dtaibi, g.venters}@unibz.it

[2] Pervasive Computing Department, Tampere University of Technology, Tampere, Finland
davide.taibi@tut.fi

[3] SASAbus, 39100 Bolzano-Bozen, Italy
windegger@sasabz.it

Abstract. *Context*: Unhandled code exceptions are often the cause of a drop in the number of users. In the highly competitive market of Android apps, users commonly stop using applications when they find some problem generated by unhandled exceptions. This is often reflected in a negative comment in the Google Play Store and developers are usually not able to reproduce the issue reported by the end users because of a lack of information.

Objective: In this work, we present an industrial case study aimed at prioritizing the removal of bugs related to uncaught exceptions. Therefore, we (1) analyzed crash reports of an Android application developed by a public transportation company, (2) classified uncaught exceptions that caused the crashes; (3) prioritized the exceptions according to their impact on users.

Results: The analysis of the exceptions showed that seven exceptions generated 70% of the overall errors and that it was possible to solve more than 50% of the exceptions-related issues by fixing just six Java classes. Moreover, as a side result, we discovered that the exceptions were highly correlated with two code smells, namely "Spaghetti Code" and "Swiss Army Knife". The results of this study helped the company understand how to better focus their limited maintenance effort. Additionally, the adopted process can be beneficial for any Android developer in understanding how to prioritize the maintenance effort.

Keywords: Continuous monitoring · Software quality · Technical debt

1 Introduction

In the quickly evolving world of mobile applications, users can nowadays choose among a large variety of alternatives. On the one hand, this huge market boosts the competition among similar apps. On the other hand, users can easily switch to competitor applications if some features are missing or – as commonly happens – not responding correctly. The cost of acquiring lost customers is always high, and losing customers because of bugs and issues – which are often related to code exceptions – occurs very frequently.

Android applications sometimes crash or are irresponsive, and several users commonly report this with negative comments in the Google Play Store, complaining about non-functioning parts such as unresponsive buttons or other issues. This behavior is often due to uncaught exceptions happening [23, 24]. We consider uncaught exceptions those exceptions not caught in "try/catch" blocks that may occur at any time due to programming errors that the developers did not expect to deal with (e.g., "OutOfMemoryError", "NullPointerExceptions") [17].

In this paper, we report on an industrial case study about an Android application developed by SASAbus SpA AG, an Italian public transportation company[1]. Since SASAbus had limited resources available for the maintenance of their application, our aim was to support them in the identification of better ways to invest their maintenance effort in order to prevent the greatest possible number of issues related to code exceptions and to have them lose the lowest possible number of customers. Therefore, our goal was to understand which code exception(s) generated most of the crashes, or generally what the bugs in the application were, to allow focusing corrective maintenance on solving the most relevant errors with the little resources available.

Crash reports are collected directly from the Google Play Store, which provides a system that collects for the developers the latest exceptions that generated a fatal error of the application.

The results of our study showed that six uncaught exceptions occurred in more than 70% of the crashes and that it is possible to reduce this high percentage by investing a limited amount of effort. Furthermore, from the manual inspection of the code that threw the exceptions, we observed the prominent presence of two code smells [3] ("Spaghetti Code" and "Swiss Army Knife") in similar exceptions. Moreover, this result confirms the findings of Microsoft, who reported that 50% of crashes are caused by 1% of bugs [18].

This paper is structured as follows: Sect. 2 describes the continuous exception monitoring approaches. Section 3 reports the case study. Section 4 discusses the results while Sect. 5 draws conclusions and provides an outlook on future work.

2 The Exception Monitoring System

Although the trend towards DevOps and continuous exception monitoring tools is growing constantly[2], to the best of our knowledge the literature on exception monitoring is still very limited [1, 2].

Exception monitoring tools have been introduced recently and are usually collocated in the operational side of the DevOps pipeline [19]; however, the majority of these instruments are crash reporting systems: when a fatal error in the application

[1] SASAbus. www.sasabus.it.

[2] https://trends.google.com.

happens, users get a summary message and are asked whether they are willing to report the details to the development team.

However, developers get only crash-related data, while non-fatal errors, which are often triggered by unhandled exceptions, remain unreported, hence not getting analyzed and solved. Recently, some tools aimed at dealing with this issue have been introduced on the market: Most of them offer ways to integrate exception-collecting plugins within the client to be monitored, while others simply require the addition of dependencies or libraries to the source code. When an unhandled exception occurs, some contextual data is collected on the client and forwarded to a server (on the premises or in the cloud), which stacks and categorizes the exceptions.

One of the most frequently adopted exception monitoring tools is Firebase, the default crash report provided in the Google Play Store. Firebase sends crash reports to developers, reporting a list of fatal crashes and the exceptions that caused them. Furthermore, Firebase also allows tracking uncaught exceptions, even though this feature requires instrumenting the code by adding a custom logger to the classes and substantially modifying all the source code.

One of the goals of our research is to improve the resolution of such problems quickly and promptly by capturing also concealed problems related to exceptions in the software, categorizing them, and solving the issues accordingly.

In Table 1, we present some of the most commonly adopted exception monitoring tools on the market.

Table 1. The most common exception management tools

Tool	Supported programming language	Exception categorization	Open source	Link
OverOps	Java, Scala, Closure, .NET	Yes	No	www.overops.com
Airbrake	All major ones	Yes	Yes	http://airbrake.io
Sentry	All major ones	Yes	Yes	www.sentry.io
Rollbar	All major ones	Yes	Yes	www.rollbar.com
Raygun	All major ones	Yes	Yes	www.raygun.org
Honeybadger	All major ones	Yes	Yes	www.honeybadger.io
Stackhunter	Java	Yes	No	www.stackhunter.io
Bugsnag	All major ones	Yes	Yes	www.bugsnag.com
Exceptionless	All major ones	Yes	Yes	www.exceptionless.com
Firebase	Android, iOS	Yes	No	https://firebase.google.com

3 The Case Study

In this section, we report on our case study, which aimed at (1) understanding how to manage and capture exceptions, and which exceptions need to be focused on in order to solve most of the errors, and (2) at reducing the frequency of exceptions.

In the following sub-sections, we will present the study process we adopted: First, we will describe the study goal as well as the questions and metrics to be collected; then we will describe the design and the execution of the study; and finally, we will present the results we obtained.

3.1 Study Goal, Questions and Metrics

According to our expectations, we formulated the goal of the case study according to the GQM approach [16] as:

Analyze the unhandled exceptions
for the purpose of prioritizing them
with respect to their frequency
from the point of view of the end user
in the context of the SASABus application

Therefore, we formulated our research questions as follows and further derived the relative metrics.

RQ1: Which are the most frequently recurring exceptions in our Android application? In this RQ, we aim at identifying the most frequent exceptions in order to rank them based on frequency. We expect to have a subset of frequent exceptions that generate most of the errors.

M1: Total number of exceptions. This metric considers all the exceptions that occurred in running apps.
M2: Number of unique exceptions.
M3: Number of occurrences of the same exception.
M4: Number of occurrences of the same exception from the same class.

As an example, consider the exceptions listed in Table 2. In this case, the total number of exceptions (M1) is six. The number of unique exceptions M2 is three (E1, E2, E3), and the number of occurrences of the same exception (M3) is three for E1, two for E2, and one for E3. However, taking into account M4, we see that E1 was raised twice in class C1 and once in class C2, while exception E2 was raised twice, both times in C1.

Table 2. Example of exceptions raised by an application

Date	Exception	Class
01/01/2017 08:00	E1	C1
01/01/2017 08:30	E1	C2
01/01/2017 09:00	E1	C1
02/01/2017 09:30	E2	C1
02/01/2017 10:30	E2	C1
02/01/2017 11:30	E3	C4

RQ2: Which and how many exception-related issues should be fixed in order to reduce the number of exceptions by 70%?

With this RQ, we analyze the frequency of the exceptions and identify those exceptions that need to be addressed in order to reduce their occurrence by 70%. This RQ was required by the industrial partner, who seeks to reduce their number of unhandled exceptions and to increase user satisfaction, as well as to reduce the number of negative comments due to some unresponsive features caused by the exceptions.

As a consequence of RQ1, we expect to be able to solve 70% of the exceptions by fixing only 20% of the issues.

As an example, considering the exceptions reported in Table 1, addressing exception E1 in class C1 and E2 in class C2 would be more beneficial than solving E1 in C2 or E3 in C4.

RQ3: Which classes generate more exceptions?

This RQ aims at understanding whether some classes deserve more attention or need to be refactored.

In our example, we can identify C1 as a class worth getting the developers' attention.

3.2 The Analyzed Application

This case study is based on the analysis of the SASAbus Android application, available on the Google Play Store[3]. The application has been downloaded by nearly 50 K users and is currently used by more than 10 K users. The source code is also available as Open Source[4]. The app was developed by SASAbus with the aim of providing bus timetables, delays and locations of buses, and other travel-related information. The software is developed with an agile process, and it migrating flawlessly from an ad-hoc process to SCRUM from several years [13].

Figure 1 shows the SASAbus application on the Google Play Store.

3.3 Results

In order to answer to our research questions, we analyzed the SASAbus Android crash reports from April 2013 to March 2017 (we chose this time range because the Google Play Store does not save reports older than four years).

Before analyzing the individual result of each RQ, we confirmed the normality of the exception distribution by means of the Shaphiro-Wilk test and analyzed the descriptive statistics.

As for **RQ1**, the SASAbus application collected 1208 exceptions in four years, with an average of 0.83 exceptions per day. Peaks in the number of exceptions per day were reached during high season (e.g., Christmas and summer time), with a maximum of eight exceptions per day.

3 SASAbus on the Google Play Store. https://play.google.com/store/apps/details?id=it.sasabz.android.sasabus&hl=en.

4 SASAbus GitHub repository. https://github.com/SASAbus/SASAbus.

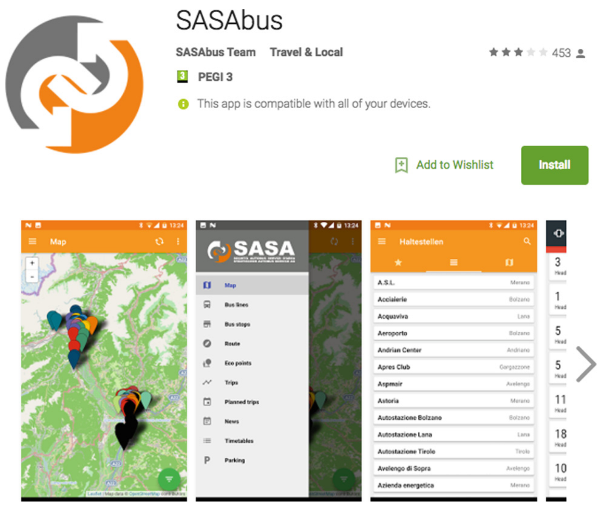

SASAbus
SASAbus Team Travel & Local
453
PEGI 3
This app is compatible with all of your devices.
Add to Wishlist
Install
Map
SASA
Haltestellen

Fig. 1. The SASAbus application on the Google Play Store

Table 3. Frequency of exceptions

Java exception	Frequency	%
java.lang.RuntimeException	433	35.8
java.lang.NullPointerException	212	17.5
java.lang.ClassCastException	124	10.3
java.lang.UnsupportedOperationException	106	8.8
java.lang.IllegalStateException	94	7.8
java.lang.ClassNotFoundException	57	4.7
android.view.WindowManager$BadTokenException	45	3.7
java.lang.IllegalAccessException	26	2.2
android.database.sqlite.SQLiteDatabaseCorruptException	23	1.9
java.lang.IndexOutOfBoundsException	23	1.9
java.lang.IllegalArgumentException	21	1.7
java.lang.InstantiationException	16	1.3
Native crash	8	0.7
java.lang.StringIndexOutOfBoundsException	7	0.6
android.database.sqlite.SQLiteException	4	0.3
java.lang.OutOfMemoryError	4	0.3
android.util.AndroidRuntimeException	3	0.2
java.io.IOException	1	0.1
java.lang.NoSuchMethodError	1	0.1
Total	1208	

We identified 19 types of unique exceptions with different frequencies (Table 3). Two exceptions were caught in more than 50% of the cases, namely "java.lang.RuntimeException" and "java.lang.NullPointerException", while other exceptions occurred rarely.

Taking into account the classes that had more exceptions, we can see that 44.8% of the exceptions were thrown by three Java classes (Table 4). However, the same exceptions were often thrown by different classes.

Table 4. Frequency of exceptions per class that raised more than 10 exceptions

Java class	Frequency	%
android.graphics.Bitmap	223	18.5
android.os.Looper	211	17.5
java.util.Collections$UnmodifiableCollection	106	8.8
it.sasabz.sasabus.ui.busstop.NextBusFragment$2	81	6.7
dalvik.system.BaseDexClassLoader	57	4.7
android.support.v4.app.Fragment	44	3.6
android.support.v4.app.FragmentManagerImpl	41	3.4
android.view.ViewRootImpl	33	2.7
it.sasabz.sasabus.opendata.client.model.BusStation	31	2.6
it.sasabz.sasabus.ui.busstop.NextBusFragment$1	31	2.6
it.sasabz.sasabus.ui.busstop.NextBusFragment$7	28	2.3
it.sasabz.android.sasabus.classes.adapter.MyWayListAdapter	27	2.2
java.lang.Class	26	2.2
android.database.sqlite.SQLiteConnection	23	1.9
android.support.v7.widget.RecyclerView$n	21	1.7
it.sasabz.sasabus.ui.busschedules.BusScheduleDetailsFragment	19	1.6
it.sasabz.sasabus.ui.busschedules.BusSchedulesFragment$4	17	1.4
it.sasabz.sasabus.ui.routing.SearchResultsFragment	17	1.4
it.sasabz.sasabus.ui.busstop.NextBusFragment$6	16	1.3
java.lang.reflect.Constructor	16	1.3
android.app.LoadedApk	13	1.1
android.view.ViewRoot	12	1
it.sasabz.android.sasabus.classes.services.CheckUpdate	11	0.9
it.sasabz.android.sasabus.fcm.a.e	10	0.8
org.mapsforge.map.reader.MapDatabase	10	0.8

The analysis of **RQ2** and **RQ3** led us to the identification of the exceptions that should be addressed by the developers. As requested by SASAbus, our goal was to identify the classes that generated 70% of the exceptions.

Table 5 reports the frequency of the exceptions per class, where the "Freq." column reports the number of times the exception was thrown by the class reported in the column "Java Class", while "Rel Freq." reports how often an exception of the same type was thrown by the selected class. For reasons of space, Table 5 only reports the

Table 5. Frequency of exceptions per class. Package named is omitted for reasons of space.

Java classes	Exception	Freq.	Rel freq.	Abs. freq.
Bitmap	RuntimeException	222	51%	18%
Looper	RuntimeException	211	49%	17%
NextBusFragment	ClassCastException	124	100%	10%
Collections $UnmodifiableCollection	UnsupportedOperationException	106	100%	9%
BaseDexClassLoader	ClassNotFoundException	57	100%	5%
NextBusFragment	NullPointerException	55	26%	5%
MyWayListAdapter	NullPointerException	54	25%	4%
BusScheduleDetailsFragment	NullPointerException	38	18%	3%
SearchResultsFragment	NullPointerException	34	16%	3%
BusStation	NullPointerException	31	15%	3%
Class	IllegalAccessException	26	100%	2%
Constructor	InstantiationException	16	100%	1%

name of the class, omitting the package. Package names can be easily associated from Table 4. As an example, "RuntimeException" was thrown 433 times, (222 times in class "Bitmap.java" and 211 times in class "Looper.java"). Finally, column "Abs Freq." reports the impact of the exception over the total number of occurrences. For instance, RuntimeException, thrown 222 times in the class Bitmap, accounted for 18% of the 1208 exceptions.

Summing up, the first seven exceptions occurred in nearly 70% of the cases. Furthermore, one method of the class "NextBusFragment" threw 124 "ClassCastExceptions" and 55 "NullPointerExceptions". Hence, focusing corrective maintenance on the first six classes of this table would fix the issues that generated 70% of the exceptions.

4 Discussion

A first analysis of the source code forced us to consider the importance of adopting a continuous exception monitoring system. In the last four years of this code's evolution, 32 classes generated a total of 1208 exceptions, and just six of them threw nearly 70% of the exceptions. Maintenance is an expensive task [4] and, considering the limited resources available to SASAbus, they will now be able to focus their maintenance effort especially on the most critical issues, solving the biggest number of issues with the lowest possible effort.

Based on the results obtained, we manually inspected the source code of these six classes and found interesting side results.

One of the main side results is that in most of the classes generating exceptions, we identified code smells, a set of structural software characteristics that indicate code or design problems that can make software hard to evolve and maintain [3].

Both of the classes that generated “RuntimeException” have a “Spaghetti Code” structure, while the other classes were generally very complex or structured as a “Swiss Army Knife”. Although this result is not statistically significant and was not obtained with a rigorous method, we are planning to train SASAbus developers in code smells and code quality practices to help them write cleaner code, which would be easier to maintain in the future. However, the result can be already beneficial from the development team, that could start to create a knowledge base of their development practices, reporting the bad practices that generated the exception. As example, we recommended to start keeping track of the data with the Agile Experience Factory [5].

Based on the results provided by this work, SASAbus decided to adopt an exception monitoring tool. Starting in May 2017, they began using Sentry because of its open source license, which allows on-premise installations to collect the data, and because of the ease of configuration, which requires only the addition of one dependency library to the code base without the need to instrument the source code.

The results of this work contribute to understanding the importance of a continuous exception monitoring approach.

As for threats to validity, in our work we prioritized the exceptions based on their frequency, since they were all related to crash reports that caused the shutdown of the application. A continuous monitoring approach that reports uncaught exceptions should also be able to classify them, i.e., not only provide their frequency, but also relate them to their severity. The results obtained are useful for SASAbus but not statistically significant, since they were not obtained using a rigorous method. These results and the approach we adopted can be validated in future studies, also applying defect prediction techniques.

5 Conclusion

In this work, we presented an industrial case study aimed at understanding the most frequent exceptions raised by an Android application adopted by more than 10 K users.

We analyzed the crash reports and the exceptions from the last four years, identifying seven types of exceptions that occurred in more than 70% of the cases.

A manual code inspection enabled us to understand that most of the exceptions were generated by bad code practices, mainly code smells, which decrease the readability, understandability, and maintainability of the code, hence causing bugs and issues [7, 12].

The results of this work helped SASAbus to clearly understand where to focus their corrective maintenance effort, working on the most frequent issues, as well as solving most of the problems with the least effort possible. Before the analysis of their exceptions, the application crashed nearly once a day because of uncaught exceptions, and several users reported the unresponsive features on the Google Play Store, such as unresponsive buttons or other exception-related issues. The results of the newly installed continuous monitoring system will be visible in the near future, when developers will be able to not only detect the most frequent exceptions, but also to intervene based on the severity of the issue.

The main lesson learned from this work is that, thanks to a continuous monitoring tool and statistical analysis of the exceptions, companies can easily understand how to better address part of the corrective maintenance effort.

A classic fault prediction approach based on code analysis, might have indicated the high possibility of running into problems when delivering the app in a proactive way. Therefore, we are planning to apply prediction models, both based on internal and on external metrics, similar to these we already applied in [6, 9, 10, 14, 15] paired with this approach.

We are currently working on analyzing change proneness in classes that generated issues, including bugs and fixes manually reported by users and developers in the issue tracker and not actually related to exceptions [12].

As described in our previous work [8, 11], future work will include the analysis of static metrics and code smells of this application, as well as understanding possible correlations between code smells and issues. Another future work will be the analysis of comments and the relations between negative comments, crashes and exception, so as to better address maintenance activities. Moreover, other approach for the automatic discover of android crashes such as [20, 21, 22] could be adopted to better focus the maintenance activities.

References

1. Krall, A., Probst, M.: Monitors and exceptions: how to implement Java efficiently. Concur. Pract. Exp. **10**(11–13), 837–850 (1998)
2. Turner, L.D., Owhoso, V.: Use ERP internal control exception reports to monitor and improve controls. Manag. Account. Q. **10**(3), 41–50 (2009)
3. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
4. Lenarduzzi, V., Sillitti, A., Taibi, D.: Analyzing forty years of software maintenance models. In: International Conference on Software Engineering (ICSE 2017), Buenos Aires, Argentina (2017)
5. Taibi, D., Lenarduzzi, V., Diebold, P., Lunesu, I.: Operationalizing the experience factory for effort estimation in agile processes. In: International Conference on Evaluation and Assessment in Software Engineering (EASE 2017), Karlskrona, Sweden, 15–16 June 2017. http://doi.org/10.1145/3084226.3084240
6. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Predicting OSS trustworthiness on the basis of elementary code assessment. In: International Symposium on Empirical Software Engineering and Measurement (ESEM 2010), Bolzano-Bozen, Italy, 16–17 September 2010. http://doi.org/10.1145/1852786.1852834
7. Taibi, D., Janes, A., Lenarduzzi, V.: Towards a lean approach to reduce code smells injection: an empirical study. In: Sharp, H., Hall, T. (eds.) XP 2016. LNBIP, vol. 251, pp. 300–304. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33515-5_30
8. Lenarduzzi, V., Stan, A.C., Taibi, D., Tosi, D., Venters, G.: A dynamical quality model to continuously monitor software maintenance. In: 11th European Conference on Information Systems Management (ECISM 2017), Genoa, Italy, 14–15 September 2017
9. Tosi, D., Lavazza, L., Morasca, S., Taibi, D.: On the definition of dynamic software measures. In: International Symposium on Empirical Software Engineering and Measurement (ESEM 2012), pp. 39–48. ACM, New York (2012). http://doi.org/10.1145/2372251.2372259

10. Taibi, D., Lavazza, L., Morasca, S., Tosi, D.: An empirical investigation of perceived reliability of open source Java programs. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012), Riva del Garda, Italy, 26–30 March 2012. http://doi.org/10.1145/2245276.2231951
11. Taibi, D., Janes, A., Lenarduzzi, V.: How developers perceive code smells and antipatterns in source code: a replicated study. Inf. Softw. Technol. J. (IST) **92**, 223–235 (2017). https://doi.org/10.1016/j.infsof.2017.08.008
12. Janes, A., Lenarduzzi, V., Stan, A.C.: A continuous software quality monitoring approach for small and medium enterprises. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (ICPE 2017 Companion), L'Aquila, Italy (2017). http://doi.org/10.1145/3053600.3053618
13. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Applying SCRUM in an OSS development process: an empirical evaluation. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (eds.) XP 2010. LNBIP, vol. 48, pp. 147–159. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13054-0_11
14. Bianco, V.D., Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: The QualiSPo approach to OSS product quality evaluation. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, pp. 23–28. ACM, New York (2010). http://doi.org/10.1145/1833272.1833277
15. Morasca, S., Taibi, D., Tosi, D.: Towards certifying the testing process of open-source software: new challenges or old methodologies? In: Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, FLOSS 2009, pp. 25–30 (2009). http://doi.org/10.1109/FLOSS.2009.5071356
16. Caldiera, G., Rombach, H.D., Basili, V.: Goal question metric approach. In: Encyclopedia of Software Engineering, pp. 528–532. Wiley, New York (1994)
17. Java Oracle: Uncaught Exceptions Documentation. https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.UncaughtExceptionHandler.html. Accessed May 2017
18. Rooney, P.: Microsoft's CEO: 80-20 Rule Applies To Bugs, Not Just Features. CRN News, 03 October 2002. http://www.crn.com/news/security/18821726/microsofts-ceo-80-20-rule-applies-to-bugs-not-just-features.htm
19. Delgado, N., Gates, A.Q., Roach, S.: A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans. Softw. Eng. **30**(12), 859–872 (2004)
20. Moran, K., Linares-Vásquez, M., Bernal-Cárdenas, C., Vendome, C., Poshyvanyk, D.: Automatically discovering, reporting and reproducing android application crashes. In: 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST). IEEE (2016)
21. White, M., Linares-Vásquez, M., Johnson, P., Bernal-Cárdenas, C., Poshyvanyk, D.: Generating reproducible and replayable bug reports from android application crashes. In: 2015 IEEE 23rd International Conference on Program Comprehension (ICPC). IEEE (2015)
22. Agarwal, S., Mahajan, R., Zheng, A., Bahl, V.: Diagnosing mobile applications in the wild. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM (2010)
23. Kechagia, M., Spinellis, D.: Undocumented and unchecked: exceptions that spell trouble. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM (2014)
24. Cinque, M., Cotroneo, D., Testa, A.: A logging framework for the on-line failure analysis of android smart phones. In: Proceedings of the 1st European Workshop on AppRoaches to MObiquiTous Resilience. ACM (2012)

Experimentation in Software Engineering

Evaluation of an Integrated Tool Environment for Experimentation in DSL Engineering

Florian Häser, Michael Felderer(✉), and Ruth Breu

Department of Computer Science, University of Innsbruck, Innsbruck, Austria
`{florian.haeser,michael.felderer,ruth.breu}@uibk.ac.at`

Abstract. Domain specific languages (DSL) are a popular means for providing customized solutions to a certain problem domain. So far, however, language workbenches lack sufficient built-in features in providing decision support when it comes to language design and improvement. Controlled experiments can provide data-driven decision support for both, researchers and language engineers, for comparing different languages or language features. This paper provides an evaluation of an integrated end-to-end tool environment for performing controlled experiments in DSL engineering. The experimentation environment is presented by a running example from engineering domain specific languages for acceptance testing. The tool is built on and integrated into the Meta Programming System (MPS) language workbench. For each step of an experiment the language engineer is supported by suitable DSLs and tools all within the MPS platform. The evaluation, from the viewpoint of the experiments subject, is based on the technology acceptance model (TAM). Results reveal that the subjects found the DSL experimentation environment intuitive and easy to use.

Keywords: Domain specific languages · DSL · Language engineering
Experimentation · Model-based software engineering
Meta Programming System · Empirical software engineering

1 Introduction

Domain specific languages (DSLs) are powerful languages used to build customized solutions for certain problem domains. In model-driven software engineering they are used because they provide less complexity and enable easier communication with domain experts [1]. They are not only widely used in practice [2], but also well researched [3,4].

A recent comprehensive systematic mapping study on DSL engineering conducted by Kosar et al. [4] revealed that within the DSL community, researchers only rarely evaluate their languages. Although all steps of the language engineering process are important, researchers mostly focus on the design and implementation, rather than domain analysis and maintenance. The mapping study provides evidence which shows that DSLs are rarely validated by taking end-user feedback into account. Thus researchers assume that they developed a DSLs perfectly tailored for the domain. Similar results are provided by Carver et al. [5] and Gabriel et al. [6].

The Meta Programming System (MPS) [7], an industry-ready language workbench, supports language engineers in the design and implementation phase. This includes the definition of syntax, semantics and the creation of language specific editors. Those editors bring remarkable benefit for the language user in defining DSL constructs.

So far, language workbenches do not provide sufficient decision support when it comes to domain analysis and language adaptation during maintenance. This is, however, essential for successful language engineering. A language engineer does not always know in advance, what effects a certain adaptation will have to a DSL. For instance, he or she can not know in advance before doing some experimentation if a certain adaptation makes a language really better with respect to aspects such as creation time or resulting document length of the scripts written in the adapted DSL.

Controlled experiments [8] allow to compare different DSL features and provide data-driven decision support for both language engineers and researchers. In the last decade, the number of conducted experiments in the field of software engineering increased. This significantly increased the body of knowledge on limitations and benefits of software engineering methods and techniques [8].

As mentioned above, this is, however, not entirely true for the field of DSL engineering, where experiments are still rare [4].

We assume that one major reason for this is the missing integrated tool support which enables especially language engineers in practice but also researchers to conduct controlled experiments directly within a DSL workbench in a straightforward and methodologically sound way. Such an integrated end-to-end tool environment also enables easy replication of experiments. This is the case because with a (semi-)formal definition the experimental setting can automatically be interchanged and reproduced.

Freire et al. [9] conducted a systematic literature review on automated support for controlled experiments. They discovered that there are already methodologies and tools available, which support parts or even the entire experimentation process. Those approaches, however, are not suitable for experiments with DSLs. They often only provide a DSL to formulate experiments, which is a useful input for us. DSL engineering is not performed on the web and its empirical analysis includes not only questionnaires but also automatic measurements, as only produced and collected in a real language workbench.

This paper contributes the empirical evaluation of an integrated tool environment for performing controlled experiments in DSL engineering. The integrated tool environment itself has been presented already as tool paper at EASE'16 [10], where it received a best tool paper award, and is summarized in this paper to provide the context for the presented empirical study. We are not proposing yet another DSL suited for experimentation, instead we have integrated existing DSLs and extended the DSL engineering environment MPS to cover language experimentation and to support the maintenance phase of the language engineering process.

MPS supports a mixed notation which includes textual, tabular and graphical representation of concepts. Our environment supports the language engineer with DSL design (the experimental object in our case) and all other phases of controlled experiments. This includes planning, operation, analysis & interpretation, and presentation & package. We based our platform on already established DSLs and tools which have been all integrated into MPS.

For the planning of experiments we base our work on the established DSL ExpDSL [11]. The language is used to formulate goals and hypotheses. For statistical methods metaR [12] is reused to simplify the application of complex R-functions for non-experts like most language engineers. The experiment operation is backed up by a MPS extension able to monitor and collect metrics like creation time or document length. For performing the analysis & interpretation phase, R output is displayed in the MPS editor and raw data can be exported as comma separated values (CSV). Finally, for presentation & package a DSL for LaTeX is applied [13]. The integration of several DSLs and representation is enabled by the projectional editors of MPS. They allow the mixture of languages and alternative representations of data, such as plots. Everything together can be displayed within one single document (see Fig. 4). The full integration enables researchers and practitioners to conduct controlled experiments in an easy way. All without media breaks and with different levels of rigor.

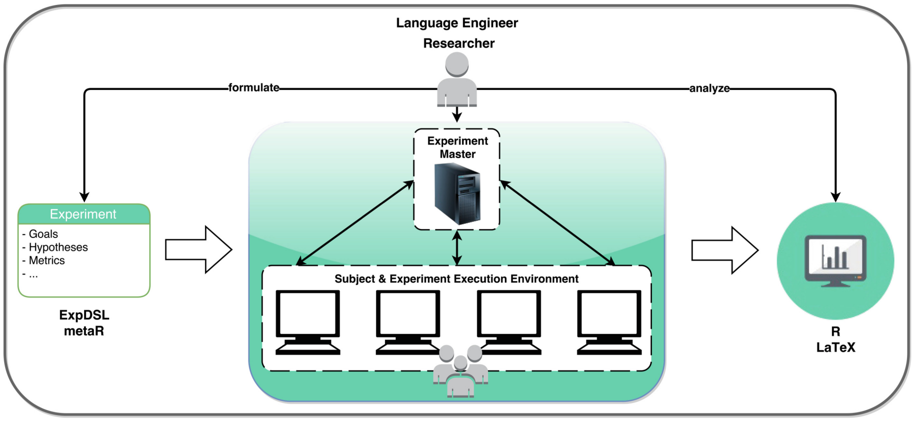


Fig. 1. Integrated DSL experiment environment architecture [10]

The remainder of this paper is organized as follows. Section 2 gives a rough overview on related approaches. Section 3 presents an example experiment to demonstrate DSL experimentation in the domain of acceptance testing of web applications. The example is used as a running example in order to explain the environment. Section 4 provides an overview of the architecture, the underlying workflow and the tool implementation of the integrated experimentation environment for DSL engineering. Sections 3 and 4 are based on a previous tool paper [10] presented at EASE'16. For better understandability the case from Sect. 3 is applied to the steps on conducting a controlled experiment. Section 5 presents a qualitative evaluation of the experiment platform. Finally, Sect. 6 concludes the paper and presents future work and visions for the tool environment.

2 Related Work

In this section we present related work with regard to evaluation of DSLs and experimentation environments for software engineering artifacts.

As already mentioned in the previous section Kosar et al. [4] did a comprehensive literature review on the DSL engineering research field. They empirically discovered that researchers are not evaluating their languages. There may be various reasons for that issue, among them missing tool support. Some recent publications provide frameworks and tools for evaluating languages before and after launching them. Barišić et al. [14] have elaborated a usability evaluation framework for DSLs. In the framework they propose the use of controlled experiments for evaluating languages. Izquierdo et al. [15] also propose that the involvement of the real end-user is essential for building and evaluating DSLs.

Tairas and Cabot [16] on the contrary presented a post-hoc evaluation analysis methodology. The DSL is analyzed after launching it based on its actual usage. It gives the language engineer insights on language artifacts that need improvement. Even though they have evaluated their approach results do not indicated if updates to the language based on the analysis also improve the language prior introducing it. We think that such an analysis cannot replace the involvement of a real user, but it is useful to identify language concepts that need improvement.

Freire et al. [9] conducted a systematic review on tool support for planning and conducting controlled experiments in general. They found that tool support is mainly limited to the planning phase. The experimenter is often supported by some sort of DSL or ontology for defining an experiment. Support means either textual as presented by Garcia et al. [17] and Siy and Wu [18] or graphically as of Cartaxo et al. [19]. The textual ontologies allow the formal definition of experiments and checking constraints regarding the validity of such.

Freire et al. [20] proposed the first integrated environment supporting all required steps of an experiment, i.e., planning, execution and interpretation. This was done by developing the ExpDSL approach.

Additionally expressibility and completeness was empirically evaluated [11] in various case studies.

Efforts in developing experiment environments have been taken by various researchers. Mostly they differ for their application domains.

In the domain of high-performance computing a set of integrated tools for supporting experiments have been developed by Hochstein et al. [21]. The 'Experiment Manager Framework' is supporting almost all experiment related activities but also has some limitations. Especially the adaptation to a different domain such as DSL engineering is difficult. For example the built-in automated data capturing and its analysis have been developed for high performance computing source code. Other limitations are regarding the lacking support for extending the platform with new statistical design types, as this is not possible easily.

Sjøberg et al. [22] developed SESE (Simula Experiment Support Environment), a web based experiment environment. Subjects are guided through the experiment with assignments. For each assignment the time to spend it, is measured. Additional material can be distributed centrally and the framework allows the centralized collection of results and generated artifacts such as source code. The solution is applicable in many different domains introduces, however, lots of media breaks, as subjects have to switch between applications and tools. As of now, automated metric capturing for language related aspects are not realizable in a web based tool. Thus, using SESE for language related experiments is not optimal.

Because of the limitations with existing approaches and the possibilities on operating within the very same language workbench a language is developed, we have decided to provide an integrated tool environment for DSL experimentation. Instead of reinventing everything, we reused different tested languages and tools. Planning an experiment is done with ExpDSL as introduced above. It forms the base template for our tool chain. ExpDSL differs from a technological point of view, as it is based on Xtext. Our environment bases on MPS.

Our DSL experimentation environment is the first one that supports all phases of experimentation in DSL engineering in one integrated environment. As mentioned before, we presented the integrated tool environment itself, which is evaluated in this paper, already in a previous publication [10]. In addition, in [23] we used the DSL engineering environment to evaluate whether support for business domain concepts is beneficial for DSLs to specify acceptance test cases. In this paper, we provide an evaluation of our integrated DSL experimentation environment based on the technology acceptance model.

3 Running Example Description

With the aid of the integrated tool environment we have performed a DSL experiment for comparing two similar testing languages for acceptance testing of web applications.

The experiment was realized with 22 graduate computer science students in the context of an exercise during a lecture. Especially, for acceptance testing, students are well suited as experimental subjects if a domain familiar to the students is chosen, because then the experiment setting with student participants is similar to the situation of acceptance testing in industry, where it is common to have acceptance tests executed by test personnel with domain knowledge, but only little experience in systematic testing, which is normally the case for students [24]. The domains for defining acceptance tests were therefore selected to be familiar to students, i.e., printing of a signed certificate of studies as well as performing a simple task on the used teaching platform.

An important requirement for the adaptation of a test language is that the time for defining a certain test case is not significantly higher after the adaptation. In reality time can be influenced by many factors such as usability issues or understanding problems. Therefore, such a metric should always be triangulated with other results, e.g., from questions of an accompanying questionnaire. Nevertheless, the analysis of creation time is well suited for describing the tool environment.

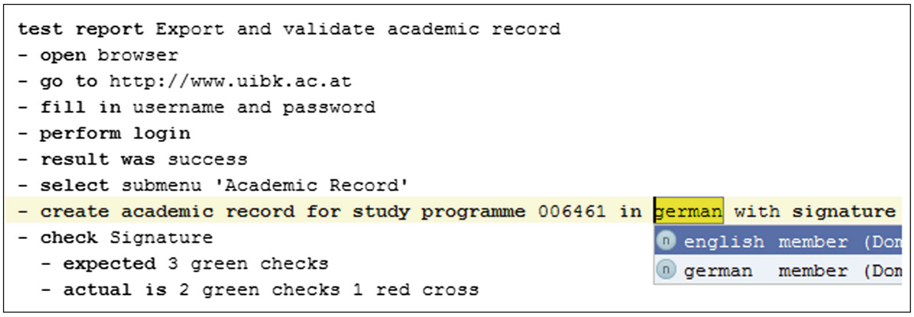
In the real experiment we have also analyzed various other metrics and factors, among them perceived quality of the created test cases.

Listing 1 shows a part of a test case written with the simple DSL (DSL_1). It only consists of two predefined keywords, the step keyword, denoted with "-", and the check keyword.

Listing 1. Example for the simplified test language

```
Test Case Specification: Participate to student
                         seminar
– open a browser
– check whether it has started successfully
– visit http://www.uibk.ac.at
– check if the site has been loaded
– click on the Link named OLAT
...
```

Figure 2 shows a test case with the extended DSL (DSL_2), in a screenshot taken during the conduction of the experiment.



```
test report Export and validate academic record
- open browser
- go to http://www.uibk.ac.at
- fill in username and password
- perform login
- result was success
- select submenu 'Academic Record'
- create academic record for study programme 006461 in german with signature
- check Signature                                      english member (Don
  - expected 3 green checks                            german  member (Don
  - actual is 2 green checks 1 red cross
```

Fig. 2. Example for the enriched test language [10]

Both languages are suitable for writing acceptance tests for web applications. DSL_1 allows the user freedom in writing test cases. It may represent a language introduced in a company for providing initial tool support in writing acceptance tests. The second language, i.e., DSL_2, is enriched with extensions from the actual application domain. It could potentially allow faster definition of test cases compared to DSL_1, but without empirical evidence the language engineer is not sure about that. DSL_2 may represent an evolution of DSL_1 for improving tool support in the creation of acceptance test cases.

Although the problem sounds simple and one could intuitively argue that using a language bound to the business domain will for sure allow a user to create faster results than without, this has not to be the case. Language users may feel restricted or confused by certain language concepts, because they are named different than expected. The conduction of a simple experiment can reveal issues with a language before introducing it by giving the language engineer data-driven decision support.

4 Integrated DSL Experimentation Environment

The architecture of the integrated tool environment is outlined in this section together with a short introduction of the methodology. Each experiment step is described with the aid of the example experiment case as presented above. The architecture and the methodology are closely aligned with the steps on planning and executing a controlled experiment as proposed by Wohlin et al. [8]. Those steps (see Fig. 3) are planning, operation, analysis & interpretation, and presentation & package.

Figure 1 shows an overview on the integrated environment. It consists of three major parts, for planning, conduction and analysis. Each part is supported by specific languages and tools. All accessible from within the language workbench.

The planning phase is supported by reusing the experiment definition language (ExpDSL) developed by Freire et al. [11]. ExpDSL was empirically evaluated in 18 experiments. In its second revision the DSL supports almost the entire experiment stack. MetaR, developed by Campagne [12] simplifies data analysis with R and makes it accessible for people that have limited experience in statistics. This might be the case for some language engineers.

In order to collect metrics during the experiment an extension to the MPS platform has been implemented. It supports the experiment operation phase in various aspects and can be configured within the language workbench.

Packaging and reporting is supported by applying the DSL for LaTeX as provided by Völter [13]. It can be used to automatically create reports and pre-filled LaTeX paper templates, which include R-plots and raw data from which one can generate PDF reports.

4.1 Methodology

In this section each supported experimentation phase is described and exemplified based on the sample experiment as introduced in Sect. 3. The methodology follows the procedure as shown in Fig. 3.

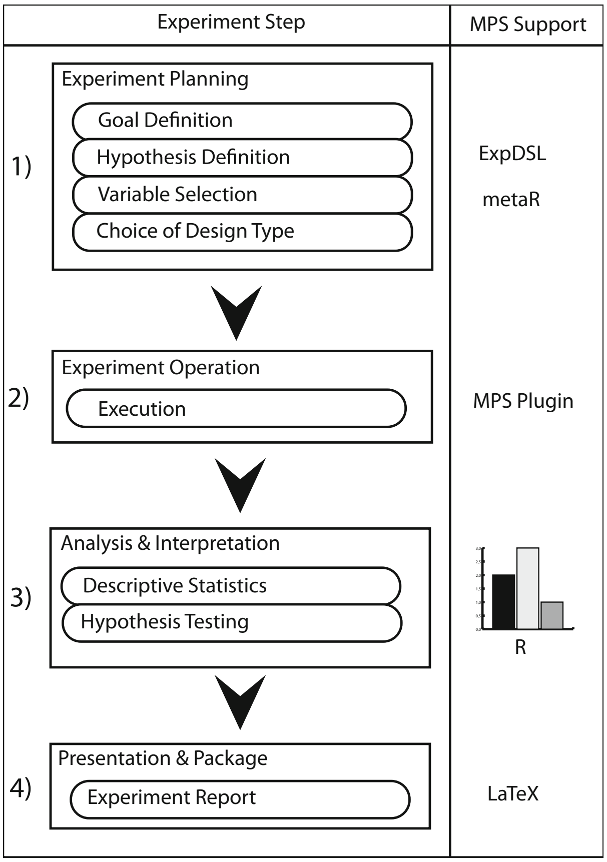


Fig. 3. Supported steps of a controlled experiment [10]

Experiment Planning. For this phase we have reused ExpDSL [11] a DSL for formulating all relevant aspects of an experiment in a precise way. It allows the researcher or language engineer formulating experiment goals, hypotheses, choose a statistical method and to specify dependent and independent variables as well as metrics in a sound way. The base language was extended with features that are based on metaR [12]. Additionally, we have introduced a mapping of certain variables to concrete data, which is collected automatically during the operation phase. This allows an automated generation of certain plots and to automatically test some of the hypotheses.

As shown in the example, some information, e.g., the abstract, is needed for generating the final report.

In practice, a language engineer may only be interested in some basic results of an experiment with lack of full rigor. The basic results are sometimes enough to provide sufficient decision support. Because of that and due to the space limitation we omit details of the complete statistical analysis like power analysis, which are for sure also possible, in the sample experiment description.

The basic example case describes the goals and hypotheses (see Listing 2) together with its alternative. Additionally, Fig. 4 sketches the related hypothesis tests and box plots. In the DSL relevant keywords are highlighted, e.g., 'Goals' or 'Hypotheses'. Additionally the goal question metric template has been implemented [25]. The actual experiment design formulation is a major aspect for the experiment conduction. One not only has to specify statistical methods, but also select variables and their corresponding metrics.

Listing 2. Example goals and hypotheses

```
Abstract This paper presents a controlled experiment...
Goals
G1: Analyze the efficiency of similar test DSLs
for the purpose of evaluation
with respect to creation time
from the point of view of a DSL user
in the context of graduate students using
assisting editors for test case creation.
Hypotheses
H0: The time to create tests with both DSLs is equal
H1: The time to create tests differs significantly
```

Some of the metrics can directly be linked to some of the predefined measures. Such a measure could be for instance the document length, the creation time or also the number of deletions within one language concept. This is enabled because the experiment platform is fully integrated to the actual language workbench.

It is important to mention that as of now, the prototype only supports 'single factor two level' experiment designs. For the upcoming milestone we planned to allow other design types and additional automated language related measurements.

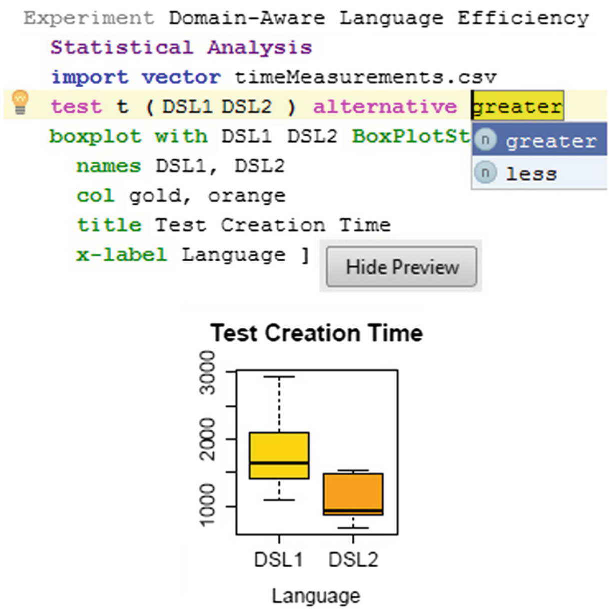
Experiment Operation. In the operation phase, the very same language workbench, used to develop the language and for planning the experiment can be used to conduct the experiment. Basically it has two operation modes, either slave or master (see Fig. 1).

After starting an experiment, test subjects request their assignment from the master instance. While solving the assignment, each client instance collects empirical evidence and sends them back to the master at the end of the experiment. For instance in our case the clients are used by the subjects for creating test cases with either DSL_1 or DSL_2. Among them time measurements, number of deletions and edits.

For supporting the operation phase and enabling the automated collection of data, we have developed an extension plug-in to the MPS language workbench. Each variable as defined in the planning phase is linked to a certain trigger. Start and stop buttons for measuring time in an ongoing task is for example one of those triggers.

The master instance (see Fig. 1) knows the formal experiment description is responsible for distributing assignments and collecting all results. After finishing the experiment, the execution of previously specified statistical analysis procedures is started and a preliminary report is shown. As shown in Fig. 4 it is a modifiable report, which can be adapted to the needs of the language engineer.

Analysis and Interpretation. The analysis & interpretation phase deals with evaluating the hypothesis given collected data. In our case for instance, after performing a normality test, a t-test can be applied in order to verify if the time to create test cases differs significantly.



```
Experiment Domain-Aware Language Efficiency
  Statistical Analysis
  import vector timeMeasurements.csv
  test t ( DSL1 DSL2 ) alternative greater
  boxplot with DSL1 DSL2 BoxPlotSt
    names DSL1, DSL2
    col gold, orange
    title Test Creation Time
    x-label Language ]
```

Fig. 4. Editor within the language workbench with instant preview [10]

Figure 4 shows an example output, that has been generated during the execution of the example experiment as introduced in Sect. 3.

During the analysis one could discover interesting insights. The ability to adapt the analysis on the fly, all assisted by the tool is one of the benefits of having an integrated tool environment.

Figure 4 shows a box plot comparing the creation times of a certain test case for DSL_1 and DSL_2 respectively.

The box-plot already indicates, that the creation time is smaller for DSL_2. For being absolutely sure and verify the significance, one needs to perform further statistical analysis. After performing a normality test, a t-test can be executed. Figure 4 shows the editor right in the moment of the definition of the test. The platform is a very initial prototype supporting a small set of statistical methods. Thus, for detailed analysis there is the possibility to export the raw data as comma separated values (CSV) file. RStudio [26] or RapidMiner [27] allow the import of such data for further analysis.

While using the experiment platform in a real experiment situation, with automated collection of language related metrics, which is important for DSL experimentation, we discovered that there is also a strong need for supporting questionnaires and analysis support.

Questionnaires are not supported as of now, but listed as a top priority feature for the upcoming release.

Presentation & Package. The final phase of the experimentation process is the creation of the report according to to experiment definition. By now the tool environment is able to automatically generate a simple PDF report with information in an easy to read way. Reports are generated from LaTeX templates which base on the recommendations of Wohlin et al. [8]. They contain all relevant information of the experiment. We are aware that this phase consists of more than just a PDF report. Collected data, both raw and generated through analysis, should be included as well as experiment protocols. Until now the prototype is only able to export raw data.

5 Evaluation of the Platform

The integrated experiment platform was used and evaluated in concrete DSL experiments as described in the previous section. In this section we provide an overview of the design of the case study which accompanied the DSL experiments. In the design and reporting of this case study, we follow the recommended guidelines for conducting and reporting case study research in software engineering by Runeson et al. [28].

As already mentioned above we have used the experiment platform for conducting a real DSL experiment. We have found significant differences between two different versions of a language. The actual implementation of the experiment, with the dedicated platform, did not produce any unconsidered issues. The success of the experiment shows already that the tool is useful and suitable for DSL experiments from the viewpoint of a language engineer, as it provides data-driven and evidence-based decision support. Experiments, however, also rely a lot on the commitment of subjects. If they do not find the tool useful, it may bias the outcome and success of the DSL experiment. Therefore we decided to further evaluate the platform from the experiments subject point of view.

The evaluation itself was inspired by the Technology Acceptance Model (TAM) [29]. TAM is a convenient theory used to explain whether, and how, users come to accept new technologies. Basically, it is used to measure the two metrics perceived usefulness and ease of use. Perceived usefulness is defined as

[29] "the degree to which a person believes that using a particular system would enhance his or her job performance". Perceived ease of use is described as [29] "the degree to which a person believes that using a particular system would be free of effort".

The subjects of the experiment platform are not going to use the system in their daily job, therefore above mentioned initial definitions are not entirely true for the sake of our evaluation. However both, usefulness and ease of use, as perceived by subjects, can influence the result of an actual DSL experiment. In order to mitigate that risk and keep the bias for the DSL experiment low we have designed the case study as follows.

5.1 Evaluation Design

Based on the goal, question, metric (GQM) template [8,30], we can formulate the goal of this experiment as follows:

In this case study, we investigate whether the domain specific language experiment platform is useful and easy to use with respect to perceived usefulness and perceived ease of use.

Research Questions. From the above formulated GQM template we have identified the following research questions:

- **RQ1:** Is the integrated tool environment useful for creating test case specifications (perceived usefulness)?
- **RQ2:** Is the integrated tool environment easy and intuitive (perceived ease of use)?

Case and Subject Selection. We have developed the experimentation platform for DSL engineering in order to conduct an experiment, in which we compared two different DSLs with respect to various aspects, such as creation time or document length. As part of the DSL experiment, we distributed a questionnaire to the subjects consisting of both quantitative and qualitative questions. Each of the experiment participants was therefore not only a subject in the DSL experiment, but also in this evaluation. Intentionally, the subjects have not been trained before. They only followed a five minute tutorial, where the basic commands of the language editor were demonstrated.

Data Collection Procedure. For the data collection procedure, we prepared a questionnaire that was distributed to the experiment subjects after completing the DSL experiment. There was no time limit for filling out the questionnaire. The questionnaire was filled out on paper and collected at the end of the experiment.

Analysis Procedure. The analysis was performed by two researchers with a mix of qualitative and quantitative methods.

In order to answer both research questions based on answers of the questionnaire, the following concepts have been evaluated as perceived by the experiment subjects. To guarantee that each subject has the same understanding of those concepts, each of them was explained shortly as follows:

To answer RQ1 on perceived usefulness, we based our questionnaire on the work of Davis [31], which elaborated a set of questions, that allow the quantification of perceived usefulness. As already mentioned above perceived usefulness, as originally defined, does not directly apply in our context. Subjects are not going to use the system or tool as part of their daily work. Therefore, we have selected and adapted the questions accordingly (see below). Note that we have not included all TAM related questions. The evaluation is only inspired by TAM.

For answering RQ2 on the perceived ease of use, we based our questionnaire on the System Usability Scale as worked out by Brooke [32]. Again we did not use the entire catalog of questions to keep this side experiment manageable. Similarly to the questions for RQ1, we only did selected those, we thought were relevant for evaluating the ease of use, of the experiment platform.

We are aware, that those adaptation make a comparison to similarly evaluated technology difficult. The resulting trend, however, gives a good indication of both perceived usefulness and ease of use.

- **RQ1.** Perceived usefulness
  - **Support**: Did you feel supported, compared to performing the task in a text editor?
  - **Benefit**: Did you get beneficial support while performing the experiment?
  - **Responsiveness**: Did you have to wait a lot for the tool, while using it?
  - **Restrictiveness**: Did you feel restricted by the tool during the experiment?
- **RQ2.** Perceived ease of use
  - **Intuition**: How quickly do you think other computer scientists need to learn to use this tool?
  - **Usability**: Was the tool in general easy to use?

In the questionnaire, subjects were able to rate above introduced concepts on a four-point Likert-Scale ranging from “Strongly agree” (1) to “Not at all” (4).

For triangulating results from the quantitative evaluation and to get a broader view and understanding on the usability of the experiment platform, we provided a set of open questions. The questions are very general, allowing a lot of freedom and creativity in answering them.

- What did you like about the experiment platform?
- What did you not like about the experiment platform?
- Do you have any suggestions for improvements for the tool?

Answers to those questions have been examined together by two of the authors and after then analyzed using the open coding technique as of Glaser and Strauss [33]. The results to this analysis are presented in Sect. 5.2.

Validity Procedure. Following the guidelines of Runeson et al. [28], the threats to validity of the evaluation have been analyzed according to construct, internal and external validity. In addition, the reliability of the evaluation was also considered. For each type of validity, we have prepared countermeasures. Countermeasures for each validity type are described in detail in Sect. 5.3.

5.2 Results and Discussion

This section presents the results of the evaluation grouped according to the two research questions as introduced in Sect. 5.1. The questions are addressing the perceived usefulness (RQ1) and the perceived ease of use (RQ2). For each of the two research questions, the results are summarized and interpreted. Extreme deviations are discussed in detail.

A total of 22 responses have been collected. As part of the validity procedure and for finding correlations, we have included questions about prior testing experience and DSLs in the questionnaire. Similar to the other quantitative questions subject had to choose from a four-point Likert-Scale. Figure 5 shows the basic statistics on the responses of those questions.

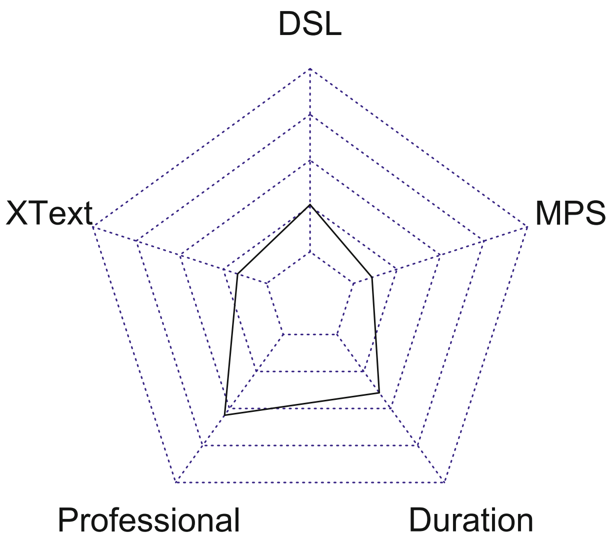


Fig. 5. Prior experience on testing and DSLs

Perceived Usefulness (RQ1). Figure 6 shows the results for the perceived usefulness criteria in form of a violin plot, which is a combination of box plot and density chart.

Subjects were able to answer the questions, as presented in Sect. 5.1, to predefined answers based on a Likert-scale ranging from “Strongly agree” (1) to “Not at all” (4).

For all criteria, the standard deviation was below or around 1, which is an indicator for meaningful results.

The two criteria “Beneficing” and “Restrictiveness” have been perceived as minimal useful. For the latter, this is a desired result, as we wanted the platform support to be as minimalistic as possible. This is good, for keeping variables that are not under the experimenters’ control, low.

Initially, we had no explanation for the bad results for the criteria “Beneficing”, however, triangulation with open coding revealed the cause. Subjects felt well-supported in the beginning of the experiment, however, they wished, for example to dynamically introduce new language concepts during the experiment. This was definitely out of scope of the experiment, and not intended at all by the experimenters.

The responsiveness of the tool was considered good.

“Support” by the experiment platform, compared to performing the exact same experiment with a regular text editor was evaluated on average as very good. These results indicate, that the participating subjects see the usefulness of the platform, leading to a high perceived usefulness in general.

Perceived Ease of Use (RQ2). The results from the questions as of RQ2 are represented in Fig. 7. Both criteria show that the use of the experiment platform was both intuitive and also easy to use, resulting in a high overall ease of use.

Similarly to the results of RQ1, the standard deviation is very low, in this case for both around 0.6. This indicates meaningful and representative results. Triangulation with open coding as presented later in this section, endorses the results.

Having a good ease of use reduces the learning phase during a DSL experiment and may lower the bias produced by it. In addition, as a great majority of the participating subjects found the environment easy to use, metrics such as time are more accurate and representing the actual time of solving a task, including only little overhead produced by the learning of how to use the tool.

For analyzing the results of the open questions and triangulating the to the quantitative ones, we used the open coding technique as proposed by Glaser and Strauss [33]. Open coding is a repetitive process, in which data is interpretively broken down analytically and assigned to codes. The codes do not belong to any existing theory, they are just based on the actual meaning of the underlying data. This technique is used to get new insights by interpreting phenomena as reflected in data in a non-standard way.

In the following, the open codes, its properties and examples of the participants’ exact words are presented for each question as described in Sect. 5.1.

For the question “What did you like about the experiment platform?” we have created the following open codes:

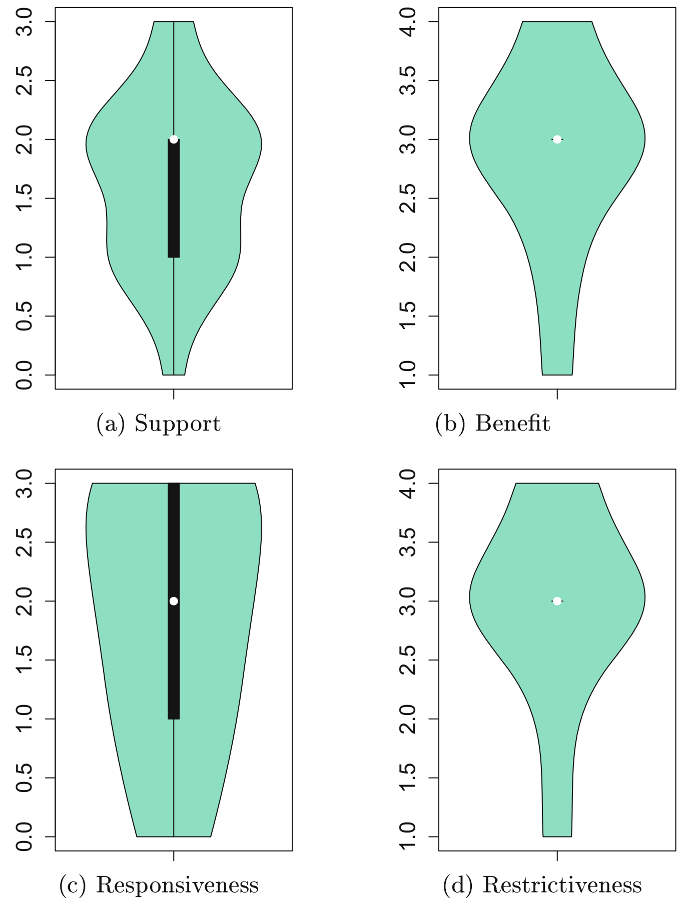


Fig. 6. Results for perceived usefulness criteria

Complexity reducing:

– Even though there were many steps, it was easy to do
– Process is very simple
– Enforces clear structure with a restricted syntax
– The tool provides a good structure, however, it is restrictive

No preliminary knowledge needed:

– Every DSL user should be able to perform the task
– No need to learn the language first
– You only need to know the CRTL+Space shortcut
– Non-computer scientists can learn quickly the tool
– This can be used by a non computer science person as well

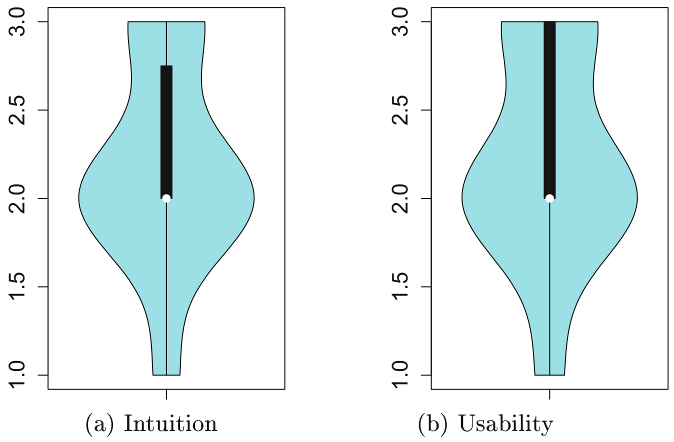


Fig. 7. Results for perceived ease of use criteria

Simple user interface:

- Simple & easy
- Each step is performed in a simple way
- Intuitive user interface
- Easy to understand

Intuitive usage:

- Straight forward
- The tool is easy to understand and to use
- Each step was intuitive
- Auto-completion feature supports intuitive usage

Quick tool:

- Quickly and easy to write steps
- Fast fluid work-flow

As one can immediately see, results from the open coding coincide with the results of the quantitative analysis. Subjects find the tool very intuitive and easy to use. One additional interesting result, which was not explicitly part of the quantitative analysis, is whether a subject needs some preliminary knowledge prior to performing the experiment task. Open coding revealed that the participants like the intuitive usage and stated that even non-computer scientists can use the tool without learning it before.

Results from the quantitative analysis indicate that the subjects felt restricted by the tool, performing the task. Open coding mitigates that fact. By being restrictive the tool enforces a good and clear structure and reduces the complexity of the outcome.

The answers for the questions “What did you not like about the experiment platform?” and “Do you have any suggestions for improvements for the tool?” only produced one code:

Missing feature:

– Copy&Paste not working
– “Ctrl+Backspace” not working
– No syntax highlighting for some keywords
– Some predefined commands were incomplete

Subjects mainly complained about missing features they know from other tools, or basic features such as copy and paste. This affects the expected perceived usefulness and has to be considered in future releases. In addition, participants did not like that only a small subset of the domain was implemented. Even though that was intended by the experimenter, this also negatively influences the perceived usefulness.

5.3 Threats to Validity

This section presents the threats to validity for our results together with the countermeasures that have been applied to mitigate those threats. As proposed by Runeson et al. [28], they are divided into construct validity, reliability, internal validity and external validity.

Construct validity addresses whether the study really represents answers to the research questions and whether it is consistent with the underlying intention. We wanted to analyze if subjects find the experiment platform easy to use and also useful. Those questions have not only been asked directly, i.e., “Was the tool, in general easy to use?” but were triangulated by additional indirect questions for which open coding was applied to analyze them.

Reliability addresses if the collected and analyzed data can be reproduced by others coming to the same results. It is a well-known threat to all data-centric studies. Results from the open coding analysis technique are of course highly depended on the background and perception of the researcher that are willing to reproduce the evaluation. A counteraction to this fact was that two of the authors did the open coding on their own. After then the results have been compared. We did not find any crucial differences during the comparison, which suggests, that others may come to the same results. In addition, we did prepare a personality test based on the Big Five [34]. It consisted of five questions all related to the conscientiousness. As a result of the test, which was very positive on average, we can ensure the validity of both the qualitative and the quantitative questions. For further increasing the validity, the test was provided in the mother tongue of the subjects, in order to minimize understanding issues.

Internal validity concerns that an observation between treatment and outcome is causal and not as desired a result of factors that are not under the experimenters control. We have considered that information exchange among participants may threaten the internal validity. Therefore we have communicated, that the participants had to work on their own without communicating to other participants in order to mitigate this threat.

External validity is concerned with the fact to what extent it may be possible to generalize the findings. It is a known issue with student experiments that they may threaten the external validity. According to Tichy [35] there are four situations that allow the use of students as subject: (1) if they are well trained for performing the task they are supposed to fulfill, (2) when comparing different methods, the trend of the difference is to be expected to be comparable to that of professionals, (3) in order to eliminate hypotheses, for example if a student experiment reveals no effect on an observable, it this is also very likely to have the same outcome with professionals, or (4) as a prerequisite for experiments with professionals. For the context of our evaluation situations 1–3 apply. This makes it very likely that the subject selection does not influence the external validity. The results can therefore to a reasonable extent be interpreted in a similar way to those of a comparable study performed in an industrial setting. For further lowering the external validity we may have to replicate the evaluation. We think that because of the advantages of projectional over textual editors it may be difficult to generalize those findings also to other language workbenches such as Xtext.

6 Conclusion

This paper presented a fully integrated tool environment for experimentation in DSL engineering based on the Meta Programming System (MPS) language workbench. All steps of experimentation, i.e., planning, operation, analysis & interpretation and presentation & packaging, are supported within the very same environment. Each step is supported by a specialized DSL implemented in MPS or by an MPS extensions, e.g., for automated metrics collection. A language engineer or a researcher is supported in performing experiments on DSLs by providing immediate, evidence-based decision support. The environment is demonstrated with a running example, where two DSLs for acceptance testing of web applications are evaluated with respect to test case creation time. The example shows the practical application of the integrated environment. To the best of our knowledge our approach is the first one providing end-to-end support for experiments in DSL engineering within a single integrated tool environment.

The application of the platform in a real DSL experiment, revealed interesting significant differences between two languages used for testing web applications. This actual application shows already the usefulness from the viewpoint of an experimenter, therefore we decided to empirically evaluate it from the viewpoint of experiment subjects. This not only because the commitment of subjects is essential for experiments. If the tool has a good perceived usefulness and ease of use we can expect a low bias created in learning how to use the tool.

Based on our initial findings we can say that the subjects had, in general, no problems with using the experiment platform in a real DSL experimentation situation. This is not only true for subjects, that had prior experience with DSLs in general or tools such as Xtext or MPS, but also for unexperienced participants. Results from the qualitatively evaluated questions, with open coding analysis, substantiate those finding. Both ease of use and usefulness are showing a positive trend, which may indicate, that the experimentation platform has only little influence on the result of the DSL experiment. However, this finding has to be further evaluated empirically, for instance by conducting case studies and interviews with DSL engineers or other users of our experimentation platform.

We already planned the extension of our tool by integrating the entire feature set as described in ExpDSL. This includes for example the definition and execution of questionnaires. Additionally, one important lesson we have learned during the conduction of the real experiment as described in this paper is the fact that the automatically collected metrics need to be triangulated with qualitative analysis. Open and closed coding analysis appeared to be a suitable analysis method for that issue, thus we plan to implement support for it in the upcoming release. Furthermore, we also plan to conduct an experiment in an industrial setting for improving the tool environment and for evaluating it from the perspective of professionals.

Acknowledgment. This research was partially funded by the research project MOBSTECO (FWF P26194).

References

1. Fowler, M.: Domain-Specific Languages. Addison-Wesley Signature Series (Fowler). Pearson Education, London (2010)
2. Völter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C.L., Visser, E., Wachsmuth, G.: DSL Engineering - Designing, Implementing and Using Domain-Specific Languages (2013). dslbook.org
3. Dias-Neto, A.C., Travassos, G.H.: Model-based testing approaches selection for software projects. Inf. Softw. Technol. **51**(11), 1487–1504 (2009)
4. Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: a systematic mapping study. Inf. Softw. Technol. **71**, 77–91 (2016)
5. Carver, J.C., Syriani, E., Gray, J.: Assessing the frequency of empirical evaluation in software modeling research. In: EESSMod (2011)
6. Gabriel, P., Goulao, M., Amaral, V.: Do software languages engineers evaluate their languages? arXiv preprint arXiv:1109.6794 (2011)
7. JetBrains Team: MPS: Meta Programming System. Jetbrains (2015)
8. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Science & Business Media, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
9. Freire, M.A., da Costa, D.A., Neto, E.C., Medeiros, T., Kulesza, U., Aranha, E., Soares, S.: Automated support for controlled experiments in software engineering: a systematic review. In: The 25th International Conference on Software Engineering and Knowledge Engineering, Boston, MA, USA, 27–29 June 2013, pp. 504–509 (2013)

10. Häser, F., Felderer, M., Breu, R.: An integrated tool environment for experimentation in domain specific language engineering. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, p. 20. ACM (2016)
11. Freire, M., Kulesza, U., Aranha, E., Nery, G., Costa, D., Jedlitschka, A., Campos, E., Acuña, S.T., Gómez, M.N.: Assessing and evolving a domain specific language for formalizing software engineering experiments: an empirical study. Int. J. Softw. Eng. Knowl. Eng. **24**(10), 1509–1531 (2014)
12. Campagne, F.: MetaR: a DSL for statistical analysis. Campagne Laboratory (2015)
13. Völter, M.: Preliminary experience of using mbeddr for developing embedded software. In: Tagungsband des Dagstuhl-Workshops, p. 73 (2014)
14. Barišić, A., Amaral, V., Goulão, M., Barroca, B.: Evaluating the usability of domain-specific languages. In: Recent Developments, Formal and Practical Aspects of Domain-Specific Languages (2012)
15. Izquierdo, J.L.C., Cabot, J., López-Fernández, J.J., Cuadrado, J.S., Guerra, E., de Lara, J.: Engaging end-users in the collaborative development of domain-specific modelling languages. In: Luo, Y. (ed.) CDVE 2013. LNCS, vol. 8091, pp. 101–110. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40840-3_16
16. Tairas, R., Cabot, J.: Corpus-based analysis of domain-specific languages. Softw. Syst. Model. **14**(2), 889–904 (2015)
17. Garcia, R.E., Höhn, E.N., Barbosa, E.F., Maldonado, J.C.: An ontology for controlled experiments on software engineering. In: SEKE, pp. 685–690 (2008)
18. Siy, H., Wu, Y.: An ontology to support empirical studies in software engineering. In: 2009 International Conference on Computing, Engineering and Information, ICC 2009, pp. 12–15. IEEE (2009)
19. Cartaxo, B., Costa, I., Abrantes, D., Santos, A., Soares, S., Garcia, V.: ESEML: empirical software engineering modeling language. In: Proceedings of the 2012 Workshop on Domain-Specific Modeling, pp. 55–60. ACM (2012)
20. Freire, M., Accioly, P., Sizílio, G., Campos Neto, E., Kulesza, U., Aranha, E., Borba, P.: A model-driven approach to specifying and monitoring controlled experiments in software engineering. In: Heidrich, J., Oivo, M., Jedlitschka, A., Baldassarre, M.T. (eds.) PROFES 2013. LNCS, vol. 7983, pp. 65–79. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39259-7_8
21. Hochstein, L., Nakamura, T., Shull, F., Zazworka, N., Basili, V.R., Zelkowitz, M.V.: An environment for conducting families of software engineering experiments. Adv. Comput. **74**, 175–200 (2008)
22. Sjøberg, D.I., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanovic, A., Koren, E.F., Vokác, M.: Conducting realistic experiments in software engineering. In: 2002 Proceedings 2002 International Symposium on Empirical Software Engineering, pp. 17–26. IEEE (2002)
23. Häser, F., Felderer, M., Breu, R.: Is business domain language support beneficial for creating test case specifications: a controlled experiment. Inf. Softw. Technol. **79**, 52–62 (2016)
24. Felderer, M., Herrmann, A.: Manual test case derivation from uml activity diagrams and state machines: a controlled experiment. Inf. Softw. Technol. **61**, 1–15 (2015)
25. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (GQM) approach. Encycl. Softw. Eng. (2002)
26. RStudio Team: RStudio: Integrated Development Environment for R. RStudio Inc., Boston, MA (2015)
27. RapidMiner Team: Rapid-I: RapidMiner: Predictive Analytics Platform (2015)

28. Runeson, P., Host, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering: Guidelines and Examples. Wiley, Hoboken (2012)
29. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Q. **13**(3), 319–340 (1989)
30. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering, vol. 93. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5
31. Davis, F.D.: User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. Int. J. Man Mach. Stud. **38**(3), 475–487 (1993)
32. Brooke, J.: SUS-a quick and dirty usability scale. Usability Eval. Ind. **189**(194), 4–7 (1996)
33. Glaser, B.G., Strauss, A.L.: The Discovery of Grounded Theory: Strategies for Qualitative Research. Transaction Publishers, Piscataway (2009)
34. Rammstedt, B., John, O.P.: Measuring personality in one minute or less: a 10-item short version of the big five inventory in English and German. J. Res. Pers. **41**(1), 203–212 (2007)
35. Tichy, W.F.: Hints for reviewing empirical work in software engineering. Empirical Softw. Eng. **5**(4), 309–312 (2000)

Smart Environments

Leveraging Smart Environments for Runtime Resources Management

Paolo Barsocchi, Antonello Calabró(✉), Francesca Lonetti, Eda Marchetti, and Filippo Palumbo

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche (CNR), via G. Moruzzi 1, 56124 Pisa, Italy
{paolo.barsocchi,antonello.calabro,francesca.lonetti, eda.marchetti,filippo.palumbo}@isti.cnr.it

Abstract. Smart environments (SE) have gained widespread attention due to their flexible integration into everyday life. Applications leveraging the smart environments rely on regular exchange of critical information and need accurate models for monitoring and controlling the SE behavior. Different rules are usually specified and centralized for correlating sensor data, as well as managing the resources and regulating the access to them, thus avoiding security flaws. In this paper, we propose a dynamic and flexible infrastructure able to perform runtime resources' management by decoupling the different levels of SE control rules. This allows to simplify their continuous updating and improvement, thus reducing the maintenance effort. The proposed solution integrates low cost wireless technologies and can be easily extended to include other possible existing equipments. A first validation of the proposed infrastructure on a case study is also presented.

Keywords: Smart environment · Monitoring · Sensors
Access control policy

1 Introduction

The interaction between people and ubiquitous computing [1], nowadays identified as a Smart Environment (SE), is an important area of interest. The SE paradigm depends on communication and cooperation among numerous devices, sensor networks embedded in the environment itself, servers in a fixed infrastructure and the increasing number of mobile devices carried by people. Thousands of smart devices are now operating in cities, gathering information and providing smart applications for e.g. environmental monitoring, energy management, traffic management, smart buildings and smart parking [2]. These devices integrate computation, networking, and physical processes and are able to monitor and control physical objects providing an extremely efficient and economic mean for improving the quality of life of citizens.

From the networking point of view, one of the solutions adopted by the SE is constituted by the Wireless Sensor Networks (WSNs) [3], which are autonomous devices managing sensing, computing and wireless communication capabilities. The integration of computation, networking, and physical processes require regular exchange of critical information in timely and reliable fashion and a specific modeling and control of the SE behavior. Considering, in particular, the control point of view, a SE usually involves three levels of rules: (i) the rules for managing and correlating sensors data and technologies; (ii) the rules able to guide the SE process, to define the users and the systems behavior, and to protect against possible problems and inconveniences faults; (iii) the access control rules that manage the resources (sensors, technologies, data, and so on) and protect against possible malicious use or security flaws.

However, independently of the formalism adopted, writing such kind of rules is a hard, verbose and error-prone activity. Considering in particular behavioral and access control ones, their modifications and updating may cause inconsistencies and/or security flaws; therefore, an accurate, time and effort consuming validation and verification should be implemented [4,5].

Especially in large scale organizations, in order to partially solve this problem, the common practice is to exploit the internal regulations and network specification requirements so to define just the basic control rules that may remain unchanged for a considerable amount of time. Thus existing solutions, generally try to adopt a static specification of the different rules and to centralize their control inside the architecture. As side effect the management of SE behavior could become quickly outdated over time, leading either inconsistencies with the evolved behavioral and technological organization environment, or security vulnerabilities.

From the previous considerations, the solution proposed in this paper relies on the decoupling the different levels of control rules, so to maximize their effectiveness and reduce as much as possible their maintenance and updating effort. The control levels considered are: the *Sensors Rules*, which define the sensors behavior and activities; the *Usage Control Rules*, which define the users and sensors interactions; and the *Access Control Rules*, which manage the accesses to the different resources expressed through a specific control policy formalism. As better detailed in the remaining of this paper, from a practical point of view, each of the control rule levels is in charge of a different independent reasoner, in order to completely separate the continuous behavioral and accesses control from physical network control.

The paper is organized as in the following. Section 2 highlights more the motivations of the proposal and the research questions considered. Section 3 presents some basic concepts about usage and access control systems. Section 4 presents the proposed infrastructure for runtime management and control of smart environments. Section 5 provides a first validation of the proposed approach on a case study. Related work are presented in Sect. 6 whereas Sect. 7 concludes the paper also hinting at future work.

2 Motivations and Research Questions

Much of the research work has been done in the field of smart buildings, smart home, smart city, however there is still the necessity of improving their quality and performance in the management of control rules (Sensors Rules, Usage Control Rules and Access Control Rules) so to make them smarter in taking intelligent and prompt decisions [6]. In the field of rule based systems, some smart proposals have been developed, especially in processing schemes [7], which mainly consist of algorithms and the complex event processing mechanism. However for reducing the overall delivery, operation and maintenance effort of the control rules typically a static approach is adopted: i.e. fixing (or rarely varying) the control rules and directly embedding them into the processing engine controlling the environment so to minimize their verification and assessment.

The target of this paper is enhance the flexibility and adaptability of the current state of the practice, by proposing a solution in which the different control rules or (subsets of them) can be activated/inhibited or on-the-fly defined and modified according to runtime organization exigencies. The idea is to leave the freedom to the organization manager to select or define the sets of control rules more suitable for his/her purpose at any specific time. For this the processing engine controlling the smart environment is enhanced with features for either directly specifying the proper set of control rules or to select the most suitable ones from a pre-defined collection of frequently adopted rules. A dedicated engine will manage the frequent updates/modifications of the set of rules, checking their correctness and compliance, and overriding when necessary the those anymore valid, without an impact on the overall management of organization.

The adoption of automated infrastructure for the definition and assessments of control rules is a valid help in their specification and implementation. Moreover it represents an important improvement for quality and performance of the smart environment, because it enhances the control of the possible violations or security problems. Summarizing the proposal of this paper will improve the current state of the practice by:

- providing the possibility to modify and implement in extremely short time, specific and temporary rules. For instance manager will have the possibility to automatically change the access control rules of a specific environment many times during a day (or particular period), in relation to either the role of the subject requiring the access, or the access time or the sensor values available for the considered environment.
- Forcing the persons in charge of control rules to formally define them for any critical situations. This avoids the specification of generic control rules which could be inefficient or even useless in case of problematic conditions. This will have several positive impacts: (i) focus on control rules on specific problems, so minimize misunderstanding or weaknesses in their implementation and management; (ii) provide more suitable solutions due to the possibility to early define or adapt the corrective actions to be performed in case of problems, security or safety flaws; (iii) better document the organization control

behavior in case of specific (critical) situations. Indeed many times specific control rules are just best practices inside an organization and not formally defined. Exploiting automatic facility for the definition and collection of control rules provides a useful knowledge database that can be used in case of similar situations or for training new personnel.
– The separation of the different control rules into three levels (Sensors Rules, Usage Control Rules and Access Control Rules) so to better map the distribution of knowledge between organization stakeholders. The use of a unique infrastructure for the definition, management and implementation of the three different levels of control rules lets the personnel with different, and many time separated, background knowledge to easily interact and work together in a unique environment. This from one side will keep the separation of roles and knowledge and from the other will improve the overall organization control correctness.

In the developing of the proposed infrastructure the following research questions have been considered:

– RQ1: Flexibility: is the proposal useful for improving the definition of control rules for different specific situations?
– RQ2: Adaptability: is the proposal useful for improving the selection of the proper control rules depending on specific environmental conditions?
– RQ3: Low cost: is the overhead introduced by the proposed infrastructure acceptable? In particular, we will assess whether it can have an impact on the overall cost of smart environment implementation, installation and integration.
– RQ4: Usability: is the proposed usable by not domain experts? In particular, we will assess whether the required technical baseline knowledge can have an impact on usability of the proposed implementation.

3 Access and Usage Control

Access control is one of the most adopted security mechanisms for the protection of resources and data against unauthorized, malicious or improper usage or modification. In the last decades, a large variety of policies and policy models have been proposed to define authorized operations on specific resources, such as (RBAC) Model [8] and (XACML) [9]. An XACML policy defines the access control requirements of a protected system. An access control request aims at accessing a protected resource in a given system whose access is regulated by a security policy. The request is evaluated on the Policy Decision Point (PDP) against the policy and the access is granted or denied.

A simplified vision of an XACML policy has a hierarchical structure: at the top level there is the policy set, which can contain in turn one (or more) policy set(s) or policy elements. A policy set (a policy) consists of a target, a set of rules and a rule combining algorithm. The target specifies the subjects, resources, actions and environments on which a policy can be applied. If a request satisfies the target of the policy set (policy), then the set of rules of the policy set (policy) is checked, else the policy set (policy) is skipped. A rule is composed by: a target, which specifies the constraints of the request that are applicable to the rule; a condition, which is a boolean function evaluated when the request is applicable to the rule. If the condition is evaluated to true, the result of the rule evaluation is the rule effect (*Permit or Deny*), otherwise a *NotApplicable* result is given. If an error occurs during the application of a request to the policy, *Indeterminate* is returned. The rule combining algorithm specifies the approach to be adopted to compute the decision result of a policy when more than one rule may be applicable to a given request. Listing 2 provides an example of XACML policy.

Usage control model (UCON) [10] is one of the emerging and comprehensive attribute based access control models that has the ability of monitoring the continuous updates in a system addressing the limitations related to attribute mutability and continuous usage permission validation. UCON model is an extension of the traditional access control models, which besides authorizations introduces new factors in the decision process, namely obligations, conditions, and mutable attributes. Mutable attributes are paired with subjects and objects, and their values are updated as a consequence of the decision process. Hence, the attributes that have been evaluated by the security policy to initially grant an access to a resource could change their values, while the access is in progress in such a way that the access right does not hold anymore. In this case, the access should be interrupted to preserve the system security. For this reason, UCON policies specify whether or not a decision factor must be evaluated before and/or during the usage of the resource (continuous policy enforcement).

4 Infrastructure

In this Section, we provide some details about the infrastructure used to decouple Sensors, Usage Control and Access Control rules in order to reply also to research questions of Sect. 2.

As shown in Fig. 1, the Infrastructure is conceptually divided in different nodes (see Fig. 1): (i) the *Access Control Engine* is the node in charge of implementing the access control management. (ii) the *Glimpse: Monitoring Infrastructure* is the node monitoring and enforcing the Sensors and Usage rules; (iii) the *Sensors and Actuators* are physical (hardware) components of the infrastructure: (iv) the *Management Interface* is the GUI (Graphical User Interface) through which the different rules can be defined and feedbacks and log analysis can be provided.

As in Fig. 1, the Administrators are in charge of providing the definition of the three levels of rules for the overall infrastructure. This can be done by means of a *GUI* on which *Rules editor* and *Policies editor* components are running. Specifically, through *Rules Editor* the Administrators can define the Sensors and Usage rules using a specific language (further details are provided in Sect. 4.2). Additionally, by means of *Policy Editor* they can define the XACML access control policies that will rule the resources access. Finally, through the *GUI*

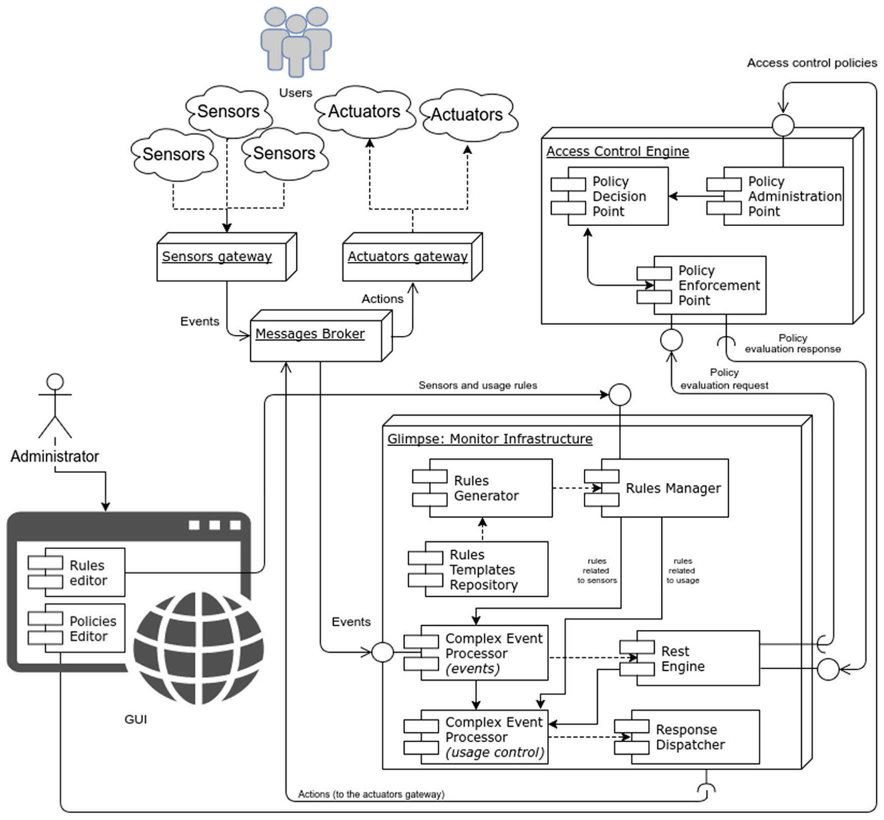


Fig. 1. Proposed architecture

the Administrators can visualize logging data, monitoring results, sensors and actuators status. In the following subsections, more details about the above mentioned nodes are provided.

4.1 Access Control Engine

This node manages the resource access by enforcing the XACML Policy defined by the Administrators. In particular, the *Access Control Engine* node contains three components (Fig. 1 top right): (i) the *Policy Enforcement Point (PEP)*, usually embedded into an application system. It receives the access request in its native format from the *Glimpse: Monitoring Infrastructure*, constructs an XACML request and sends it to the *Policy Decision Point (PDP)*; it receives the PDP responses and forwards them to the *Glimpse: Monitoring Infrastructure* through its REST (REpresentational State Transfer) Interface called *REST Engine*; (ii) the *Policy Decision Point (PDP)* evaluates the policy with respect to the request and returns the response, including the authorization decision to the *PEP*; (iii) the *Policy Administration Point (PAP)* is the component entity in charge of managing the policies and deploying them on the PDP; it receives the XACML access control policy by the *Management Interface*.

4.2 Monitoring Components

The monitor infrastructure, integrated into the proposed infrastructure, is a flexible, adaptable and dynamic solution independent of any specific sensor or access control network notation or execution. With respect to similar components and tools currently available, the monitor infrastructure included in this proposal, has been enhanced with facilities for activation the counter measures or the recovering activities in case of violations of some performance constraints. These constraints are not mandatory specified at the system startup, but can be automatically raised from the rule engines involved or can be improved at runtime by injecting new rules on the complex event processors. The monitoring framework presented in this paper has been inspired by the monitoring architecture presented in [11,12]. The *Glimpse: Monitoring Infrastructure* node (Fig. 1) manages the complex event processing and the interactions with *Sensors*, *Actuators* and *Access Control Engine*, and includes new features devoted to the usage and access control request generation.

The main monitoring components are:

- The *Rules Manager* component is in charge of orchestrating the rules generation starting from the templates stored within the component *Rule templates Repository* through the *Rules Generator* component.
- The *Rules Generator* is the component in charge of synthesizing the rules starting from the directives received by the *Rules Manager* by means of techniques based on generative programming approaches [13,14].
- The *Rules Templates Manager* is an additional internal repository storing the meta-rules enabling the run-time adaptation by means of generative procedures.
- The *CEP - Events* CEP (Complex Event Processing) Events is a rule engine realized by means of the Drools rule language [15]. It correlates the events flowing from *Sensors* with the rules loaded by the *Rules Manager* component.
- The *CEP - Usage* is in charge of correlating complex events generated by the *CEP - Events* with the rules related to the usage of the resources, loaded by the *Rules Manager*.
- The *Rest Engine*, is the component in charge of communicating through REST [16] interfaces with the *Access Control Engine* in order to send/receive the *Access Control Engine* request/response.
- The *Response Dispatcher* through the *Message Broker (AMQ)*, realized by means of ActiveMQ [17], sends events to the actuators managed by the *Actuators gateway*.

The peculiarities of the proposed architecture is to include for the first time a chain of two CEP entities, the *CEP - Events* and the *CEP - Usage*, for decoupling the activities concerning the management of the sensors from those more related to the administration of the resource usage and alarming situations. This makes easier the definition of new primitive events generated by (new/updated) sensors and the inferring of events in the form of composite events in a way completely independent of the access and usage control rules. Moreover, it lets a quick and high level updating of the general resource access and usage regulations and the planning of specific corrective actions in case of alarms or resource violations, leveraging from the specific sensor network on which they are implemented.

From a practical point of view, all the communications among monitoring components are performed through messages sent to the *Message Broker (AMQ)*, running on top of an Enterprise Service Bus like ServiceMix [18]. In order to improve the communication security, each component exposes a SSL certificate (self-signed certificates).

4.3 Sensors and Actuators Components

Sensors and *Actuators* (in top left side of Fig. 1) are deployed over the network and communicate with the infrastructure through the *Sensors gateway* and the *Actuators gateway*, respectively. These hardware components send messages to the *Glimpse: Monitoring Infrastructure* through a *Message Broker (AMQ)* using a predefined event type.

From a technical point of view, the sensor nodes considered in the proposed infrastructure are based on the module Core2530, produced by WaveShare Electronics[1]. The sensor nodes are connected with a ZigBee node[2], which is able to: evaluate the real power consumption of the target environment; identify all the indoor movements of users through Passive Infrared Sensors (PIR-based motion sensors); detect environmental noise; measure temperature and relative humidity. Every node of the distributed sensor network is configured as a ZigBee router to exploit multi-hop functionality and it periodically sends the measured data to the ZigBee coordinator. The ZigBee network ensures a reliable communications in indoor environments without suffering due to the multipath effect [19]. Moreover, each user is equipped with a Bluetooth Low Energy (BLE) beacon[3], which periodically sends a message useful for locating and identifying the user. Figure 2 shows the hardware used for sensing the environment. In particular, on the left there are the RadBeacon Dot[4] and the BLED112[5] used as a sender and receiver beacon, while on the right side there is a ZigBee node.

The middleware, named *Sensor Weaver*, uses ZB4O[6] to interact with sensors and actuators deployed in the ZigBee networks [20] and integrates the BLE in

[1] http://www.wvshare.com/.
[2] http://www.zigbee.org/.
[3] https://developer.mbed.org/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/.
[4] http://store.radiusnetworks.com/collections/all/products/radbeacon-dot.
[5] https://www.bluegiga.com/.
[6] http://zb4osgi.aaloa.org/.

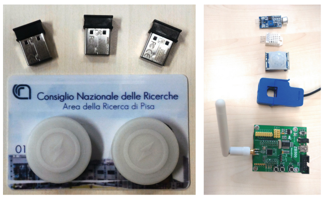

Consiglio Nazionale delle Ricerche
Area della Ricerca di Pisa
01

Fig. 2. The hardware used to sense the environment

order to abstract the different kinds of technologies. The gateway node provides access to the sensors discovered through an IP network and a communication platform. The main goal of Sensor Weaver is to provide a secure communication platform for the exchange of sensing information in a distributed sensor network environment [21,22]. Moreover, Sensor Weaver also provides tools and applications that enable long-term sensor monitoring and automatically control the actuators deployed in the WSN [23,24].

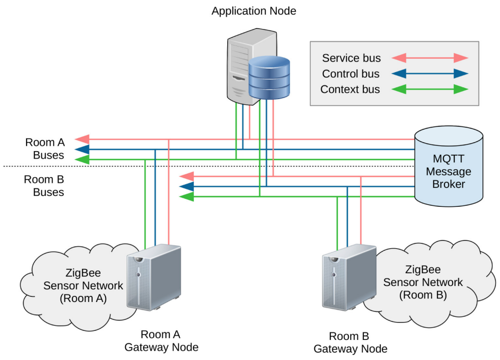


Fig. 3. The architecture of sensor weaver

In order to separate communication concerns of Sensor Weaver, we designed several communication buses. Each bus has a specific managing role (see Fig. 3): (i) a *Service Bus* for the service life-cycle events; (ii) a *Context Bus* for the sensor measurement updates; (iii) a *Control Bus* for the invocations of actuators. We implement Sensor Weaver on top of the OSGi platform and we use the MQTT messaging service [25,26].

4.4 Research Questions Analysis

As evidenced by the technical description of the infrastructure, its development has been focused on the improving as much as possible its flexibility and adaptability. In particular the availability of the *Management Interface* makes easier the definition of different kinds of control rules, lets to target on specific situations and provides the visualization of the monitor results. Moreover the availability of editors, such as the *Rules editor* and *Policies editor*, let a more friendly and usable definition of the control rules especially for people not expert in the different specification languages. All this evidences positively reply to the RQ1 and RQ4 of Sect. 2.

Concerning instead the RQ2, adaptability is guaranteed by the separation of the infrastructure into the different nodes (see Fig. 1), each one responsible of the implementation and management of separate set of control rules. The facilities provided let to detect and to correlate events generated by different layers supporting in parallel sensors, access and usage control features. In particular the innovate adoption of a chain of two CEP entities, assures the decoupling of the activities concerning the management of the sensors from those related to the resource usage. This let also a better management of alarming conditions and maximizing the adaptability of the infrastructure to different situations and exigencies.

Finally considering the RQ3, as highlighted in Sect. 4.3, cost of the proposed infrastructure is mitigated by the choice of the technology adopted for the infrastructure implementation. Indeed among the different proposals, the trade-off solution adopted in this paper relies on ZigBee [27]. This is a recognized low cost standard-based wireless technology designed to address the unique needs of low-power WSNs, and to model the different resource capabilities. It guarantees the non-invasiveness of the installations and the possibility of integration with other possible existing equipments.

5 Infrastructure Application

In this section, we describe the usage of the proposed infrastructure for the management of a cold storage room inside the *ISTI-CNR* (Istituto Scienza e Tecnologie dell'Informazione - Consiglio Nazionale delle Ricerche of Pisa Italy)[7] research area. However due to space limitation sand for aim of simplicity, we report here just a simplified description of the management of this medical cold storage called *Laboratory Cold Room (LCR)*. It is out of the scope of this paper to go into the complex details of the sets of rules necessary for managing the LCR. Here, we voluntarily keep the scenario simplified to better explain the role and the advantages of the proposed infrastructure. The complete description of the implementation can be found in [28].

The control of Laboratory Cold Room focuses on three different main aspects: to keep the temperature required for safe and secure storage of a wide range of laboratory materials; to rule the access to the room; to activate corrective actions in case of detected violations or alarming situations. For this the room has been instrumented with different sensors and several sets of control rules have been defined. These last include: (i) Sensor Rules for managing the security boundary value of each sensor or combination of them (for instance, the tolerance

[7] http://www.isti.cnr.it.

temperature, humidity ranges, and so on); (ii) Access Control policies ruling who and when can access LCR (name or the role of people that are allowed to work in the LCR); (iii) Usage Control Rules for managing sensors failures, resource violations, and alarming situation in general (for instance, the technical personnel to be called in case of problems).

5.1 Case Study Set up

As shown in Fig. 4 the Laboratory Cold Room has been equipped with a beacon receiver for gathering data from BLE beacons and with several sensors (see Sect. 4.3 for more details), which are: Temperature and humidity; Presence (PIR-based); Energy consumption; Noise detector; RFID Bagde Reader for Room access Control. An instance of the *Monitoring Infrastructure* has been deployed on: *ubuntu@n037.smart-applications.area.pi.cnr.it*, a virtual machine running on top of ISTI-CNR cloud infrastructure, while the *Access Control Engine* was running on *avalon.isti.cnr.it*. The probes that generate events related to the sensors and the actuators are running on top of the middleware: *energia.isti.cnr.it* and the events are flowing through the message broker *AMQ* running on *atlantis.isti.cnr.it*.

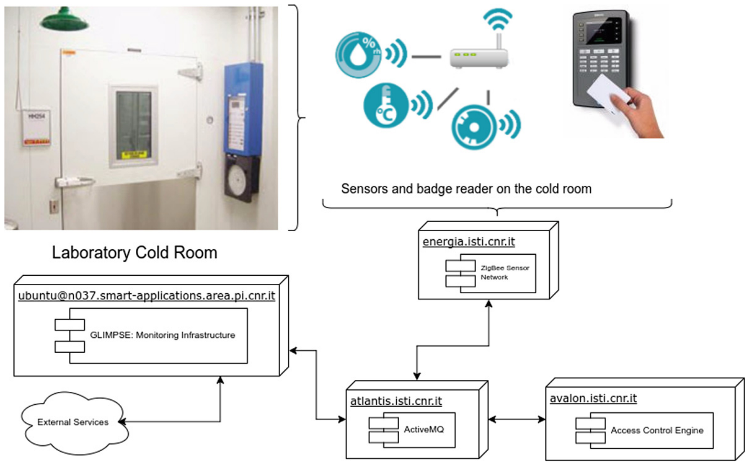


Fig. 4. Deployment configuration

5.2 Sensors, Access and Usage Rules Management

In this Section, we focus on the interaction between *CEP - Events*, *CEP - Usage* and the Access control Engine for the enforcement of the sensors, access and usage rules. Specifically, as shown in Fig. 1, through the *Rules Editor* the Administrators loaded the sensors rule useful for monitoring the sensors status and the access rules for the identification of who is currently asking resource access.

When an employee, through the RFID Bagde Reader, tries to access the LRC, the *CEP - Events* receives the access request and extracts the room ID and the badge ID. By querying the ISTI-CNR internal employees database, the *CEP - Events* retrieves (Role, Owner room id) attributes related to the user who is asking for the access. Using the data collected, the *CEP - Events* sends a *Policy evaluation request* through the *Rest Engine* to the *Access Control Engine* node and a *PdpAccessRequest* event to the *CEP - Usage* to notify that an access request has been sent. The *PEP* translates the request into an XACML access request and sends it to the *PDP*, which evaluates the request according to the access control policies injected by *PAP*, and sends back the response to the*PEP*, which in turn sends back to the *CEP - Usage* through the *Rest Engine*.

```
1   [..setup and import omitted..]
2
3   declare SensorFailureEvent
4    @idroom: int
5    @idsensor: int
6   end
7
8   rule "Check␣data␣from␣temperature␣sensor"
9     no-loop true
10    salience 1
11    dialect "java"
12  when
13    $aEvent:GlimpseBaseEventSB(this.isConsumed == false, this.isException == false,
14      (this.getTemperature == null || < -20 || > 0 ) );
15
16    $bEvent:GlimpseBaseEventSB(
17      this.isConsumed == false, this.isException == false,
18      (this.getTemperature == null || < -20 || > 0 ),
19      this after $aEvent, this.getSensorID == $aEvent.getSensorID);
20  then
21    SensorFailureEvent failureDetected = new SensorFailureEvent(idRoom,idSensor);
22    CepBinder.sendEventTo("CEP␣-␣Usage", failureDetected);
23    $aEvent.setConsumed(true);  $bEvent.setConsumed(true);
24    retract($aEvent);  retract($bEvent);
25  end
```

Listing 1. Sensors rule

An example of a sensors rule used by the *CEP - Events* for controlling all the installed sensors is shown in Listing 1. In particular, when the *CEP - Events* receives from the monitored sensors, for two consecutive times, null or out-of-range values (lines 13–14 and 17–18 of Listing 1), the *CEP - Events* generates a complex event called *SensorFailureEvent* for notifying the detected failure to the *CEP - Usage*, so that it can activate the corrective actions. For confidential reasons, we do not provide here the complete specification of the XACML access control policies adopted for managing access to the different rooms inside the ISTI-CNR research area. As reported in Listing 2, we just show an extract of some of the rules implemented in ISTI-CNR access control policies so to better explain the potentialities and features of the proposed infrastructure. Among the different types of rooms (resources) of the ISTI-CNR access control policies, here we focus on three of them: *common room, office room*, and *LCR*.

The ISTI-CNR access control policies specify different kinds of employees (subjects); however, considering the LRC, the most important are: *Biologist, Physician* and *Technician*. The policies manage also several types of actions for each room, however in this section we only consider the simpler one: the *access*.

Finally, the ISTI-CNR access control policies specify different environment values and conditions; for aim of simplicity here we only consider the case in which the environment represents the different time slots, in which an employee can access the different rooms. Considering Listing 2. the rules specify that:

1. Rule 1: each employee can access his own office and the common rooms during the business-time (from 8am to 8pm);
2. Rule 2: only an employee, who is either *Biologist* of *Physician*, can access the *LCR* during the business-time;
3. Rule 3: The *Technician* can access the *LCR* at any time.

```
<Policy PolicyId="SmartPolicy" RuleCombiningAlgId="first-applicable">
  <Target>
   <Subjects><Subject><SubjectMatch MatchId="string-equal">
      <AttributeValue DataType="string">employee</AttributeValue>
     </SubjectMatch></Subject></Subjects>
   <Resources><Resource><ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">CNR</AttributeValue>
     </ResourceMatch></Resource></Resources>
   <Actions><Action><ActionMatch MatchId="string-equal">
      <AttributeValue DataType="string">acces</AttributeValue>
     </ActionMatch></Action></Actions>
  </Target>
  <Rule RuleId="Rule1" Effect="Permit">
   <Target>
   <Resources>
    <Resource><ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">CNR</AttributeValue>
     </ResourceMatch>
     <ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">office room</AttributeValue>
     </ResourceMatch></Resource>
    <Resource><ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">CNR</AttributeValue>
     </ResourceMatch>
     <ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">common room</AttributeValue>
     </ResourceMatch></Resource>
   </Resources>
   <Environments>
    <Environment><EnvironmentMatch MatchId="time-equal">
      <AttributeValue DataType="time">8:00:00</AttributeValue>
     </EnvironmentMatch></Environment>
    <Environment><EnvironmentMatch MatchId="dayTimeDuration-equal">
      <AttributeValue DataType="dayTimeDuration">PT12H</AttributeValue>
    </EnvironmentMatch></Environment></Environments>
  </Target>
 </Rule>
 <Rule RuleId="Rule2" Effect="Permit">
  <Target>
    <Resources><Resource><ResourceMatch MatchId="string-equal">
       <AttributeValue DataType="string">LCR</AttributeValue>
      </ResourceMatch></Resource></Resources>
    <Environments>
     <Environment><EnvironmentMatch MatchId="time-equal">
       <AttributeValue DataType="time">8:00:00</AttributeValue>
      </EnvironmentMatch></Environment>
     <Environment><EnvironmentMatch MatchId="dayTimeDuration-equal">
       <AttributeValue DataType="dayTimeDuration">PT12H</AttributeValue>
       </EnvironmentMatch></Environment></Environments>
   </Target>
   <Condition><Apply FunctionId="string-at-least-one-member-of">
     <AttributeValue DataType="string">biologist</AttributeValue>
     <AttributeValue DataType="string">physician</AttributeValue>
    </Apply></Condition>
  </Rule>
  <Rule RuleId="Rule3" Effect="Permit">
   <Target>
    <Subjects><Subject><SubjectMatch MatchId="string-equal">
      <AttributeValue DataType="string">technician</AttributeValue>
    </SubjectMatch></Subject></Subjects>
    <Resources><Resource><ResourceMatch MatchId="string-equal">
      <AttributeValue DataType="string">LCR</AttributeValue>
    </ResourceMatch></Resource></Resources>
   </Target>
  </Rule>
  <Rule RuleId="default" Effect="Deny"/>
</Policy>
```

Listing 2. Smart environment access policy

At run time, the request sent by the *CEP - Events* to the *Access Control Engine* is evaluated by the PDP component and the corresponding reply is sent back to the *CEP - Usage*, which uses the received (permit or deny) response to allow the resource access or to deny it in case of possible violations or resource misuses. In both cases, the *CEP - Usage* is in charge of notifying the Actuators of the (corrective) actions to be executed. An example of usage rule implemented by the *CEP - Usage* is shown in Listing 1. In particular, the rule checks if there are pending access requests to the *LCR* and ongoing alarms. In this last case, it retrieves from the employee data base the contact data of the technician in charge of managing the alarm and it inhibits any possible access to *LCR*, apart from the selected technician.

5.3 Maintenance Activity Scenario

This section describes the management of the scenario in which an alarm is raised by the sensors and a corrective maintenance request is sent to the technician. The scenario preconditions are the following: (i) Each employee is registered on the internal ISTI-CNR personal data base; (ii) Each employee accesses the different rooms by means of a personal badge equipped with a beacon bluetoot, as shown in Fig. 2; (iii) The *LCR* room is closed by default and constantly monitored by sensors able to send events to the Monitoring Infrastructure; (iv) No one is currently inside the LCR and sensor values are within their allowed ranges.

Initially, a *Physician* requires to access the LCR by using the LCR RFID badge reader connected the to nearest network. According to the interaction described in Sect. 5.2, an event is sent to the *CEP - Events* through the *Message Broker* and the proper access request is sent to the *Access Control Engine*. This last evaluates the request and sends back the response to the *PEP*, which in turn sends back to the *CEP - Usage* through the *REST Engine*.

As shown in Listing 3, if: any revocation of permission is ongoing (line 8), there are not critical conditions (i.e. the values of temperature, humidity, energy consumption, noise are in the allowed ranges - line 18), and PDP response includes a permit (i.e. *Physician* requires to access the LCR during the business-time -line 15), the *CEP - Usage* sends an event to the *Actuator gateway* for enabling the door opening through the *Response Dispatcher*.

Supposing instead that a critical condition has been detected by *CEP - Events*, for instance either the power consumption sensors is out of range or there are significant variations in the noise or the temperature is in a not allowed range, a *SensorFailureEvent* event is sent to the *CEP - Usage* (line 46 of Listing 1). This last overrides the PDP response allowing the access to the technician only and sends an event to the *Actuator gateway* for enabling the door opening to the technician only (line 50–51 of Listing 3).

Moreover, the *CEP - Usage* sends an alarm event to the Supervision through a specific *Actuator gateway* for requesting exceptional maintenance of the LCR (line 37 of Listing 3).

```
[..setup and import omitted..]

declare SensorFailureEvent
 @idroom: int
 @idsensor: int
end

rule "If␣there␣are␣NOT␣pending␣alarm␣forward␣PDP␣access␣response"
no-loop true
salience 1
dialect "java"

when
  $aEvent:PdpAccessRequest();
  $bEvent:PdpAccessResponse(this.isConsumed == false, this.isException == false,
    this.getIdRequest == $aEvent.getIdRequest, ($bEvent.getResponse == "Permit" || "Deny"),
    this after $aEvent);
  not(SensorFailureEvent(this.isConsumed == false, this.isException == false,
    this.idRoom == $aEvent.idRoom, this.idSensor == $aEvent.idSensor));

then
  Actuators.ManageAccess($aEvent.getIdSensor(),$aEvent.getIdroom(), $bEvent.getResponse()));
end

rule "If␣there␣are␣failures␣take␣countermeasures"
no-loop true
salience 1
dialect "java"

when
  $aEvent:SensorFailureEvent();
then
  Alarm.NotifyToSupervision($aEvent.idsensor, $aEvent.idroom);
end

rule "If␣there␣are␣pending␣alarm␣check␣accesses"
no-loop true
salience 1
dialect "java"

when
  $aEvent:PdpAccessRequest();
  $bEvent:PdpAccessResponse(this.isConsumed == false, this.isException == false,
    this.getIdRequest == $aEvent.getIdRequest, ($bEvent.getResponse == "Permit" || "Deny"),
    this after $aEvent);
  $cEvent:SensorFailureEvent(this.isConsumed == false, this.isException == false,
    this.idRoom == $aEvent.idRoom);

then
  Actuators.ManageAccess($aEvent.getIdSensor(),$aEvent.getIdroom(),
            PersonnelDatabase.checkIfIsTechnician($aEvent.getIdUser));
end
```

Listing 3. Usage rule

5.4 Results Analysis and Lesson Learned

For space limitation we just provided very few details about the adoption of the proposed infrastructure inside ISTI-CNR research area. The experiment described here is part of a larger one that will involve control of all the ISTI-CNR area. The main peculiarity of the proposed approach is that the control of the different rooms is not centralized, but can be specialized time to time according the different research exigencies and in agreement with general administrative and security regulations.

Though the proposed infrastructure each lab head, or even each researcher, has the freedom to specify its own control rules depending on the sensors installed in the room or the required behavior, without changing those for the rest the area. Considering specifically the LRC, the infrastructure proposed let a detailed control management of the many PhD students requiring the access to the laboratory. Indeed without a deep impact on the generic control rules, and again in agreement with administrative and security regulations, it was possible to differentiate for each PhD student, both the allowed access time and the allowed activity when specific experimentation were ongoing.

From this experimentation two main considerations have come to light:

– Different stakeholders may have different views of the control management of a specific environment, sometimes even ignoring which are the common best practices or the corrective activities. In the specific case of LRC defining precisely the Sensors, Usage and Access control rules required many interviews and interactions with different ISTI personnel having separate competencies: researchers from one side and technicians from the other. However, the adoption of the proposed infrastructure forced them to provide, for the first time, the documentation of the procedures ruling the LRC laboratory, to highlight the critical points both from the sensors and usage point of view and to define precisely responsibilities and activities to be performed in case of security and safety flaws.
– Leaving the freedom of each lab head to define the more suitable control rules, evidenced a stringent necessity to improve the infrastructure with more dynamic features for a careful validation of consistency and correctness of the set of rules so to avoid violations of the general administrative and security regulations.

6 Related Work

This work spans over several research directions, including: smart environment, access and usage control and monitoring approaches.

Enabling Platforms in Smart Environments: The SE paradigm depends on communication and cooperation between numerous devices, sensor networks embedded in the environment itself, servers in a fixed infrastructure and the increasing number of mobile devices carried by people. In order to enable the SE paradigm, diverse platforms and software infrastructures have been proposed in the literature [29]. Among these, FI-WARE[8] is emerging as a core standard platform for Smart and Connected Communities (SCC) [30,31]. The FI-WARE project is producing new tools to facilitate the development of application and fostering a major inclusion of software standards for smart cities [32]. These tools are provided as Generic Enablers (GE): software components that can be configured and deployed on a cloud platform in order to easily implement an application. Another important enabling platform for SE is represented by the universAAL architecture[9], with a particular focus on IoT and Ambient Assisted Living (AAL) [33]. Besides its concrete open source implementation, universAAL proposes an architectural reference model based on a set of virtual communication buses as semantic brokers for context events, semantic service requests (and

[8] http://www.fiware.org.
[9] http://www.universaal.info/.

responses), and user interaction functions. For these reasons, we used the separation of concerns and the service discovery capabilities offered by the universAAL reference model to build the middleware architecture proposed in this paper.

Access and Usage Control: Concerning the testing of access control policies, combinatorial approaches have been proven to be effective in the automated generation of the test cases (access requests). Among the various proposals, the Targen tool [34] generates test inputs by using combinatorial coverage of the truth values of independent clauses of XACML policy values, while the X-CREATE tool [35] relies on combinatorial approaches of the subject, resource, action and environment values taken from the XACML policy. The main advantage of this approach with respect to Targen is the higher structural variability of the derived test inputs able to guarantee the coverage of the input domain of the XACML policy. Other works address model-based testing and provide a methodology for the generation of test cases based on combinatorial approaches of the elements of the model (role names, permission names, context names).

Concerning the Usage Control systems, the work in [36] proposes a usage control model based on UCON and describes a framework to implement it in an operating system kernel, on top of the existing DAC mechanism. Other available solutions, such as [37], propose proactive mechanisms for preventing possible policy violations and present a combination of runtime monitoring and self-adaptation to simplify the autonomic management of authorization infrastructures. In recent years, as surveyed in [38], testing of authorization systems has been focused on evidencing the appropriateness of the UCON enforcement mechanism, focusing on the performance analysis or establishing proper enforcement mechanisms by means of formal models. Finally, the authors of [39] address the testing of the Policy Decision Point (PDP) implementation within the PolPA authorization system, which enables history-based and usage-based control of accesses proposing two testing strategies specifically conceived for validating the history-based access control and the usage control functionalities of the PolPA PDP.

Monitoring: Several general-purpose monitoring proposals are currently available, which can be mainly divided into two groups: those that are embedded in the execution engine, such as [40,41], and those that can be integrated into the execution framework as an additional component, such for instance [42–44]. Both the solutions have specific advantages. For sure, an embedded solution reduces the performance delay of the execution framework, mainly in terms of interactions and communication time. Coverage indicators can be directly evaluated by the execution framework, which can also execute corrective actions in case of important deviations. The main disadvantage of these approaches is the lack of flexibility in the data collection, the coverage measure definition and the language adopted.

7 Conclusions

In this paper, we proposed an infrastructure for runtime management and control of smart environments. The main advantages of the proposed solution are its flexibility and the possibility of decoupling the different levels of rules that are defined and implemented for managing the resources of the smart environments and regulate the access to them. Specifically, three levels of rules are defined: the *Sensor Rules* for correlating sensors data and technologies; the *Usage Control Rules*, which define the users and sensors interactions; and the *Access Control Rules*, which manage the accesses to the different resources expressed through a specific control policy formalism. This allows an easy maintenance and updating of control rules when context changes or constraints violations take place.

A first validation on a real cased study, considering a medical cold storage and implementing an XACML policy, has been described. The presented scenario evidenced the effectiveness of the proposed approach to correlate events generated by different sensors and to leverage different levels of rules for raising alarms, when critical situations are detected.

As a future work, we would like to validate the proposed solution in other smart environments with different peculiarities and security constraints as well as different access control policy specification languages. Moreover, we plan to extend the infrastructure to include more refined levels of rules, further decoupling the management and control functionalities of the proposed infrastructure.

Acknowledgments. This work has been partially funded by the projects GAUSS, a National Research project (MIUR, PRIN 2015, Contract 2015KWREMX), and SIGS (FAR/FAS 2014-2016 research program of the Tuscany Region).

References

1. Weiser, M.: The computer for the 21st century. Sci. Am. **265**(3), 94–104 (1991)
2. Tragos, E.Z., Bernabe, J.B., Staudemeyer, R.C., Luis, J., Ramos, H., Fragkiadakis, A., Skarmeta, A., Nati, M., Gluhak, A.: Trusted IoT in the complex landscape of governance, security, privacy, availability and safety. In: Digitising the Industry-Internet of Things Connecting the Physical, Digital and Virtual Worlds. River Publishers Series in Communications, pp. 210–239 (2016)
3. Dargie, W., Poellabauer, C.: Fundamentals of Wireless Sensor Networks: Theory and Practice. Wiley, Hoboken (2010)
4. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E.: An automated testing framework of model-driven tools for XACML policy specification. In: 9th QUATIC 2014, Guimaraes, Portugal, 23–26 September 2014, pp. 75–84 (2014)
5. Daoudagh, S., Kateb, D.E., Lonetti, F., Marchetti, E., Mouelhi, T.: A toolchain for model-based design and testing of access control systems. In: MODELSWARD 2015 Angers, Loire Valley, France, 9–11 February 2015, pp. 411–418 (2015)
6. Ahmad, A., Rathore, M.M., Paul, A., Hong, W.H., Seo, H.: Context-aware mobile sensors for sensing discrete events in smart environment. J. Sens. (2016)
7. Drăgoicea, M., Bucur, L., Pătraşcu, M.: A service oriented simulation architecture for intelligent building management. In: Falcão e Cunha, J., Snene, M., Nóvoa, H. (eds.) IESS 2013. LNBIP, vol. 143, pp. 14–28. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36356-6_2
8. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)

9. OASIS Standard: eXtensible Access Control Markup Language (XACML) Version 2.0 (2005)
10. Park, J., Sandhu, R.: The UCON ABC usage control model. ACM Trans. Inf. Syst. Secur. (TISSEC) **7**(1), 128–174 (2004)
11. Calabrò, A., Lonetti, F., Marchetti, E.: Monitoring of business process execution based on performance indicators. In: 41st, EUROMICRO-SEAA 2015, Madeira, Portugal, 26–28 August 2015, pp. 255–258 (2015)
12. Calabrò, A., Lonetti, F., Marchetti, E.: KPI evaluation of the business process execution through event monitoring activity. In: ES 2015, Basel, Switzerland, 14–15 October 2015, pp. 169–176 (2015)
13. Czarnecki, K., Eisenecker, U.W.: Generative Programming - Methods, Tools and Applications. Addison-Wesley, Boston (2000)
14. Bertolino, A., Calabrò, A., De Angelis, G.: A generative approach for the adaptive monitoring of SLA in service choreographies. In: Daniel, F., Dolog, P., Li, Q. (eds.) ICWE 2013. LNCS, vol. 7977, pp. 408–415. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39200-9_34
15. Drools, J.: Drools Fusion: Complex Event Processor. http://www.jboss.org/drools/drools-fusion.html
16. Wilde, E., Pautasso, C. (eds.): REST: From Research to Practice. Springer, New York (2011). https://doi.org/10.1007/978-1-4419-8303-9
17. Apache: Apache ActiveMQ. http://activemq.apache.org/
18. Oracle: Java message service documentation. Technical report (2016). http://www.oracle.com/technetwork/java/docs-136352.html
19. Barsocchi, P., Oligeri, G., Potortì, F.: Measurement-based frame error model for simulating outdoor Wi-Fi networks. IEEE Trans. Wirel. Commun. **8**(3), 1154–1158 (2009)
20. Furfari, F., Girolami, M., Lenzi, S., Chessa, S.: A service-oriented zigbee gateway for smart environments. J. Ambient Intell. Smart Environ. **6**(6), 691–705 (2014)
21. Palumbo, F., Ullberg, J., Štimec, A., Furfari, F., Karlsson, L., Coradeschi, S.: Sensor network infrastructure for a home care monitoring system. Sensors **14**(3), 3833–3860 (2014)
22. Barsocchi, P., Ferro, E., Fortunati, L., Mavilia, F., Palumbo, F.: EMS@CNR: an energy monitoring sensor network infrastructure for in-building location-based services. In: 2014 International Conference on High Performance Computing & Simulation (HPCS), pp. 857–862. IEEE (2014)
23. Barbon, G., Margolis, M., Palumbo, F., Raimondi, F., Weldin, N.: Taking arduino to the internet of things: the ASIP programming model. Comput. Commun. **89**, 128–140 (2016)
24. Palumbo, F., La Rosa, D., Chessa, S.: GP-m: mobile middleware infrastructure for ambient assisted living. In: 2014 IEEE Symposium on Computers and Communication (ISCC), pp. 1–6. IEEE (2014)
25. Kim, Y., Schmid, T., Charbiwala, Z.M., Srivastava, M.B.: ViridiScope: design and implementation of a fine grained power monitoring system for homes. In: Proceedings of the 11th International Conference on Ubiquitous Computing, pp. 245–254. ACM (2009)
26. Girolami, M., Palumbo, F., Furfari, F., Chessa, S.: The integration of ZigBee with the GiraffPlus robotic framework. In: O'Grady, M.J., Vahdat-Nejad, H., Wolf, K.-H., Dragone, M., Ye, J., Röcker, C., O'Hare, G. (eds.) AmI 2013. CCIS, vol. 413, pp. 86–101. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-04406-4_10

27. Palumbo, F., Barsocchi, P., Furfari, F., Ferro, E.: AAL middleware infrastructure for green bed activity monitoring. J. Sens. **2013** (2013)
28. CNR: Smartcampus technical report. Technical report (2016). http://www.smart-applications.area.pi.cnr.it/doc/smartareaTechnicalDescription.pdf
29. Namiot, D., Sneps-Sneppe, M.: On software standards for smart cities: API or DPI. In: Proceedings of the 2014 ITU Kaleidoscope Academic Conference: Living in a Converged World-Impossible Without Standards?, pp. 169–174. IEEE (2014)
30. Glikson, A.: Fi-ware: core platform for future internet applications. In: Proceedings of the 4th Annual International Conference on Systems and Storage (2011)
31. Sun, Y., Song, H., Jara, A.J., Bie, R.: Internet of things and big data analytics for smart and connected communities. IEEE Access **4**, 766–773 (2016)
32. Ramparany, F., Marquez, F.G., Soriano, J., Elsaleh, T.: Handling smart environment devices, data and services at the semantic level with the FI-ware core platform. In: Big Data 2014, pp. 14–20. IEEE (2014)
33. Salvi, D., Montalva Colomer, J.B., Arredondo, M.T., Prazak-Aram, B., Mayer, C.: A framework for evaluating ambient assisted living technologies and the experience of the universaal project. J. Ambient Intell. Smart Environ. **7**(3), 329–352 (2015)
34. Martin, E., Xie, T.: Automated test generation for access control policies. In: Supplemental Proceedings of 17th International Symposium on Software Reliability Engineering (ISSRE), November 2006
35. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E., Schilders, L.: Automated testing of extensible access control markup language-based access control systems. IET Softw. **7**(4), 203–212 (2013)
36. Teigao, R., Maziero, C., Santin, A.: Applying a usage control model in an operating system kernel. J. Netw. Comput. Appl. **34**(4), 1342–1352 (2011)
37. Bailey, C.: Application of self-adaptive techniques to federated authorization models. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 1495–1498. IEEE (2012)
38. Nyre, Å.A.: Usage control enforcement - a survey. In: Tjoa, A.M., Quirchmayr, G., You, I., Xu, L. (eds.) CD-ARES 2011. LNCS, vol. 6908, pp. 38–49. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23300-5_4
39. Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E., Martinelli, F., Mori, P.: Testing of PolPA-based usage control systems. Softw. Q. J. **22**(2), 241–271 (2014)
40. Daoudagh, S., Lonetti, F., Marchetti, E.: Assessment of access control systems using mutation testing. In: TELERISE 2015, Florence, Italy, 18 May 2015, pp. 8–13 (2015)
41. Bertolino, A., Daoudagh, S., Kateb, D.E., Henard, C., Traon, Y.L., Lonetti, F., Marchetti, E., Mouelhi, T., Papadakis, M.: Similarity testing for access control. Inf. Softw. Technol. **58**, 355–372 (2015)
42. Carvallo, P., Cavalli, A.R., Mallouli, W., Rios, E.: Multi-cloud applications security monitoring. In: Au, M.H.A., Castiglione, A., Choo, K.-K.R., Palmieri, F., Li, K.-C. (eds.) GPC 2017. LNCS, vol. 10232, pp. 748–758. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57186-7_54
43. Bertolino, A., Calabró, A., Lonetti, F., Marchetti, E.: Towards business process execution adequacy criteria. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2016. LNBIP, vol. 238, pp. 37–48. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27033-3_3
44. Calabrò, A., Lonetti, F., Marchetti, E., Spagnolo, G.O.: Enhancing business process performance analysis through coverage-based monitoring. In: 10th QUATIC 2016, Lisbon, Portugal, 6–9 September 2016, pp. 35–43 (2016)

Author Index