Lecture Notes in Computer Science 6182

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Roel Wieringa Anne Persson (Eds.)

Requirements Engineering: Foundation for Software Quality

16th International Working Conference, REFSQ 2010
Essen, Germany, June 30–July 2, 2010
Proceedings

Springer

Volume Editors

Roel Wieringa
University of Twente
Enschede, The Netherlands
E-mail: r.j.wieringa@utwente.nl

Anne Persson
University of Skövde
Skövde, Sweden
E-mail: anne.persson@his.se

Library of Congress Control Number: 2010929494

CR Subject Classification (1998): D.2, C.2, H.4, F.3, K.6.5, D.4.6

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-642-14191-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-14191-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

This volume compiles the papers accepted for presentation at the 16th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2010), held in Essen during June 30 and July 1-2, 2010.

Since 1994, when the first REFSQ took place, requirements engineering (RE) has never ceased to be a dominant factor influencing the quality of software, systems and services. Initially started as a workshop, the REFSQ working conference series has now established itself as one of the leading international forums to discuss RE in its (many) relations to quality. It seeks reports of novel ideas and techniques that enhance the quality of RE products and processes, as well as reflections on current research and industrial RE practices. One of the most appreciated characteristics of REFSQ is that of being a highly interactive and structured event. REFSQ 2010 was no exception to this tradition.

In all, we received a healthy 57 submissions. After all submissions had been carefully assessed by three independent reviewers and went through electronic discussions, the Program Committee met and finally selected 15 top-quality full papers (13 research papers and 2 experience reports) and 7 short papers, resulting in an acceptance rate of 38 %.

The work presented at REFSQ 2009 continues to have a strong anchoring in practice with empirical investigations spanning over a wide range of application domains.

As in previous years, these proceedings serve as a record of REFSQ 2010, but also present an excellent snapshot of the state of the art of research and practice in RE. As such, we believe that they are of interest to the whole RE community, from students embarking on their PhD to experienced practitioners interested in emerging knowledge, techniques and methods. At the time of writing, REFSQ 2010 has not taken place yet. All readers who are interested in an account of the discussions that took place during the conference should consult the post-conference summary that we intend to publish as usual in the *ACM SIGSOFT Software Engineering Notes.*

REFSQ is essentially a collaborative effort. First of all, we thank Klaus Pohl for his work as General Chair of the conference. We also extend our gratitude to Ernst Sikora and Mikael Berndtsson who served REFSQ 2010 very well as Organization Chair and Publication Chair, respectively. Also we thank Andreas Gehlert for serving very well as Workshop and Poster Chair and Mikael Berndtsson for his work as Publications Chair.

As the Program Chairs of REFSQ 2010, we deeply thank the members of the Program Committee and the additional referees for their careful and timely reviews. We particularly thank those who have actively participated in the Program Committee meeting and those who have volunteered to act as shepherds to help finalize promising papers.

April 2010

Roel Wieringa
Anne Persson

REFSQ 2010 Conference Organization

General Chair

Klaus Pohl	University of Duisburg-Essen, Germany

Program Committee Co-chairs

Roel Wieringa	University of Twente, The Netherlands
Anne Persson	University of Skövde, Sweden

Organizing Chair

Ernst Sikora	University of Duisburg-Essen, Germany

Publications Chair

Mikael Berndtsson	University of Skövde, Sweden

Program Committee

Ian Alexander	Scenarioplus, UK
Aybüke Aurum	University New South Wales, Australia
Daniel M. Berry	University of Waterloo, Canada
Jürgen Börstler	University of Umeå, Sweden
Sjaak Brinkkemper	Utrecht University, The Netherlands
David Callele	University of Saskatchewan, Canada
Alan Davis	University of Colorado at Colorado Springs, USA
Eric Dubois	CRP Henri Tudor, Luxembourg
Jörg Dörr	Fraunhofer-IESE, Germany
Christof Ebert	Vector, Germany
Anthony Finkelstein	University College London, UK
Xavier Franch	Universitat Politècnica de Catalunya, Spain
Samuel Fricker	University of Zürich and Fuchs-Informatik AG, Switzerland
Vincenzo Gervasi	Università di Pisa, Italy
Martin Glinz	University of Zürich, Switzerland
Tony Gorschek	Blekinge Institute of Technology, Sweden
Olly Gotel	Independent Researcher, New York City, USA
Paul Grünbacher	University of Linz, Austria
Peter Haumer	IBM Rational, USA
Patrick Heymans	University of Namur, Belgium

Matthias Jarke	RWTH Aachen, Germany
Sara Jones	City University, London, UK
Natalia Juristo	Universidad Politécnica de Madrid, Spain
Erik Kamsties	University of Applied Sciences Dortmund, Germany
Kim Lauenroth	University of Duisburg-Essen, Germany
Søren Lauesen	IT University of Copenhagen, Denmark
Seok-Won Lee	University of North Carolina at Charlotte, USA
Nazim H. Madhavji	University of Western Ontario, Canada
Raimundas Matulevičius	University of Tartu, Estonia
Ana Moreira	Universidade Nova de Lisboa, Portugal
Haris Mouratidis	University of East London, UK
John Mylopoulos	University of Toronto, Canada
Cornelius Ncube	Bournemouth University, UK
Andreas Opdahl	University of Bergen, Norway
Barbara Paech	University of Heidelberg, Germany
Oscar Pastor	Valencia University of Technology, Spain
Gilles Perrouin	University of Luxembourg
Gil Regev	EPFL and Itecor, Switzerland
Björn Regnell	Lund University, Sweden
Colette Rolland	University of Paris-1— Panthéon Sorbonne, France
Camille Salinesi	University of Paris 1 – Panthéon Sorbonne, France
Kristian Sandahl	Linköping University, Sweden
Peter Sawyer	Lancaster University, UK
Kurt Schneider	University of Hanover, Germany
Norbert Seyff	City University, London, UK
Guttorm Sindre	NTNU, Norway
Janis Stirna	Royal Institute of Technology, Sweden
Eric Yu	University of Toronto, Canada
Didar Zowghi	University of Technology Sydney, Australia

External Reviewers

Willem Bekkers	Daniel Kerkow
Andreas Classen	Dewi Mairiza
Alexander Delater	Anshuman Saxena
Oscar Dieste	Kevin Vlaanderen
Arash Golnam	Inge van de Weerd
Florian Graf	Richard Berntsson Svensson
Wiebe Hordijk	Robert Heinrich Rumyana
Jennifer Horkow	Proynova Sebastian
Cedric Jeanneret	Barney Sira Vegas
Isabel John	

Table of Contents

Requirements Patterns

Product Families II

Requirements Engineering in Practice

Natural Language

Security Requirements

Poster

Keynote Talk
Piecing Together the Requirements Jigsaw-Puzzle

Ian Alexander

Scenarioplus Ltd., UK
iany@scenarioplus.org.uk

Software developers have been made to write requirements for their projects since the 1960s. Researchers have investigated every imaginable technique. But requirements are still not being put together well. Something is going wrong.

One reason is that while different schools of research advocate powerful methods – goal modeling, scenario analysis, rationale modeling and more – industry still believes that requirements are stand-alone imperative statements. The mismatch between the wealth of techniques known to researchers and the impoverished lists of shall-statements used in industry is striking.

The solution cannot be found by devising yet more elaborate techniques, yet more complex puzzle-pieces. Even the existing ones are scarcely used in industry. Instead, we need to work out how to assemble the set of available "puzzle-pieces" – existing ways of discovering and documenting requirements – into simple, practical industrial methods.

Another reason is that existing textbooks, and perhaps requirements education and training too, largely assume that projects are all alike, developing stand-alone software from scratch. But projects are constrained by contracts, fashion, standards and not least by existing systems. The problems they must solve, and the techniques they need to use, vary enormously. Pure and simple "green-field" development is the exception.

This talk suggests:

- what the pieces of the requirements jigsaw-puzzle are – for example, scenario analysis and goal modelling;
- how, in general, they can be fitted together – for example, as sequences of activities and by traceability;
- how, more specifically, projects of different types can re-assemble the pieces to solve their own puzzles – for example, by tailoring imposed templates, or developing processes appropriate to their domain and situation.

There are numerous answers to each of these questions. Perhaps the real message is that there is not one requirements engineering, but many.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, p. 1, 2010.

Understanding the Scope of Uncertainty in Dynamically Adaptive Systems

Kristopher Welsh and Pete Sawyer

Computing Department, Lancaster University Lancaster, UK
{welshk,sawyer}@comp.lancs.ac.uk

Abstract. [**Context and motivation**] Dynamically adaptive systems are increasingly conceived as a means to allow operation in changeable or poorly understood environments. [**Question/problem**] This can result in the selection of solution strategies based on assumptions that may not be well founded. [**Principle ideas/results**] This paper proposes the use of claims in goal models as a means to reason about likely sources of uncertainty in dynamically adaptive systems. Accepting that such claims can't be easily validated at design-time, we should instead evaluate how the system will behave if a claim is proven false by developing a validation scenario. [**Contribution**] Validation scenarios may be costly to evaluate so the approach we advocate is designed to carefully select only those claims that are less certain, or whose falsification would have serious consequences.

Keywords: self adaptation, dynamically adaptive system, goal models, uncertainty, claims.

1 Introduction

Self-adaptation is emerging as a design strategy to mitigate maintenance costs in systems where factors such as complexity, mission-criticality or remoteness make off-line adaptation impractical. Self-adaptation offers a means to respond to changes in a system's environment by sensing contextual or environmental change at run-time and adapting the behaviour of the system accordingly. In this paper we refer to self-adaptive systems as dynamically adaptive systems (DASs) to reflect their ability to adapt autonomously to changing context at run-time.

Dynamically adaptive systems have now been deployed in a number of problem domains [1] yet remain challenging to develop because there is typically a significant degree of uncertainty about the environments in which they operate. Indeed, this uncertainty is the primary reason why a DAS must be able to self-adapt; to continue to operate in a range of contexts with different requirements or requirements trade-offs. DASs remain challenging to develop, despite advances made at different levels in the software abstraction hierarchy and by communities as diverse as AI and networking. At the architectural level [2], for example, compositional adaptation [3] promotes the re-use of DAS components. Compositional adaptation is one such approach in which structural elements of the system can be combined and recombined at run-time using adaptive middleware (e.g. [4]).

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 2–16, 2010.

Despite such advances, and the seminal work of Fickas and Feather [5] in requirements monitoring, the RE community has only recently started to address the challenges of dynamic adaptation. Our own recent contribution [6, 7, 8] has been to investigate the use of goal-based techniques for reasoning about the requirements for DASs. In [7], we advocated the use of *claims* from the NFR framework [9] in i* [10] strategic rationale models to enhance the traceabilty of DAS requirements.

In this paper, we go a step further and argue for the utility of claims in DAS goal models as markers for uncertainty. This research result has emerged as an unexpected side-effect of our work on claims for tracing. Design rationale in a DAS is not always founded on good evidence, but sometimes on supposition about how the system will behave in different, hard-to predict contexts. Thus claims can serve not only as design rationale but also as proxies for analysts' understanding. It is crucial that the consequences of decisions based on assumptions that may subsequently prove to be false are understood, even if there is insufficient data to validate the assumptions themselves. We propose that a *validation scenario* should be defined to evaluate the effect of a claim turning out to be false.

The primary contribution of the paper is a simple means for reasoning about hierarchies of claims to understand how uncertainty propagates throughout these hierarchies. We show how this knowledge can be leveraged to improve the robustness of a DAS' specification using validation scenarios while minimizing the number of validation scenarios that need to be defined and evaluated. We demonstrate our approach using a case study drawn from a sensor grid that was deployed on the River Ribble in the Northwest of England. This sensor grid has acted as a preliminary evaluation for our use of claim reasoning in DASs.

The rest of the paper is structured as follows. In the next section, section 2, we introduce what we mean by claim reasoning and in section 3 we explain how an existing DAS modeling process may be adapted to include claims. We then use a case study to illustrate claim reasoning about a DAS in section 4, and conclude with a brief survey of related work (section 5) and final conclusions (section 6).

2 Claim Reasoning

In [7] we augmented i* models used to model DASs with claims, a concept borrowed from the NFR toolkit [9]. As is now well-known within RE, i* supports reasoning about systems in terms of agents, dependencies, goals and softgoals. We showed how claims can be used to record requirements traceability information by explicitly recording the rationale behind decisions, in cases where the contribution links assigned to softgoals for different solution alternatives in i* strategic rationale (SR) models don't reveal an obvious choice. We argued that this enhances the tracing information in a way that is particularly important for a system that can adapt at run-time to changing context.

As described above, claims capture the rationale for selecting one alternative design over another. As an example and before considering the effect of claims with respect to self-adaptive behaviour, consider the fragment of a simple SR model of a robot vacuum cleaner for domestic apartments depicted in Fig 1. The vacuum cleaner has a goal to clean the apartment (*clean apartment*) and two softgoals; to avoid

causing a danger to people within the house (*avoid tripping hazard*) and to be economical to run (*minimize energy costs*). The vacuum cleaner can satisfy the clean apartment goal by two different strategies that have been identified. It can clean at night or when the apartment is empty. These two strategies are represented by two alternative tasks related to the goal using means-end relationships. The choice of best strategy is unclear because at this early stage of the analysis, it is hard to discriminate between the extent to which each solution strategy satisfices the softgoals. The balance of –ve and +ve effects on satisficement of the softgoals appears to be approximately the same for both, but to different softgoals. This is depicted by the contribution links labeled *help* and *hurt*. However, the choice is resolved using a claim, which has the effect of asserting that there is *no tripping hazard.* The claim thus *breaks* the *hurt* contribution link between the task *clean at night* and the softgoal *avoid tripping hazard*. The *break*-ing claim nullifies the contribution link to which it is attached. In this case it nullifies the negative impact that night cleaning was presumed to have on tripping hazard avoidance. In turn, this has the effect of promoting the night cleaning strategy over the empty apartment cleaning strategy since it now appears to better satisfice the two softgoals. The inverse of a *break*-ing claim is a *make*-ing claim, which lends additional credence to a contribution link, implying the importance of the satisfaction of a softgoal with a *helps* link or the unacceptability of failing to satisfy a softgoal with a *hurts* link[1]. Note that claims speak of the *importance* of the effects captured by the contribution link, not the *magnitude* of the effect, which can be captured using fine-grained contribution links.

The *no tripping hazard* claim is sufficient to select the night cleaning strategy, but only if there is sufficient confidence in the claim that no hazard is offered. However, the analyst may have greater or lesser confidence in a claim, so claim confidence spans a range from axiomatic claims in which full confidence is held, to claims that are mere assumptions. At the assumption end of the claim confidence range, a claim is essentially a conjecture about a Rumsfeldian "known unknown" [11] and thus serves as a marker of something about which uncertainty exists.

If a claim is wrong, the performance of the system may be unsatisfactory, or the system may exhibit harmful emergent behaviour, or even fail completely. Ideally, the claims should be validated before the system is deployed. In a DAS, this may be very hard to do, however. Since the world in which a DAS operates is imperfectly understood, at least some conjectural claims are likely to be impossible to validate at design-time with complete assurance.

Given this fundamental limitation on claim validation, the behaviour of a system should be evaluated in cases where claims turn out to be false. To do this, a validation scenario should be designed for each claim to help establish the effects of claim falsification, such as whether it causes undesirable emergent behaviour. We do not define the form of a validation scenario; it may be a test case or some form of static reasoning. However, developing and executing validation scenarios for each claim can be expensive. A validation scenario should be developed for every possible combination of broken and unbroken claims. Hence, the number of validation scenarios (T) is $T=2^n-1$ where n represents the number of claims that make or break a softgoal contribution link. One of the three *target systems* (explained below) of the *GridStix* system

[1] Note that there are several other types of contribution and claim link to those presented here.

we describe later, would need 31 validation scenarios. Fortunately, it is possible to reduce this number by focusing attention only on claims towards the assumption end of the claim confidence range and by using *claim refinement models.*

While claims provide the rationale for selection of one solution strategy over another, the derivation of a claim may be obscure. A claim derivation is obscure if the logic of the claim is not obvious from its name. In this case, the rationale for the claim can be represented explicitly in a hierarchical claim refinement model, with the facts and assumptions from which the claim is derived also represented as claims. The claims form nodes in claim refinement model the branches of which can be AND-ed or OR-ed.

In Fig 1, the *no tripping hazard* claim is obscure because it appears as an assertion with no supporting evidence. Fig 2 is a claim refinement model that shows that the *no tripping hazard* claim is derived from three other claims: *family sleeps at night*, *vacuum is easy to see* and *vacuum has warning light*, arranged as a hierarchy of claims in a claim refinement model. Note that the root of a hierarchy of claims is at the bottom. This *bottom-level claim* is what appears on the SR diagram. The claim *vacuum is easy*

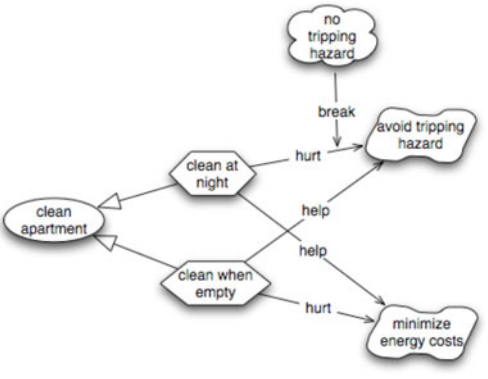


Fig. 1. A robot vacuum cleaner

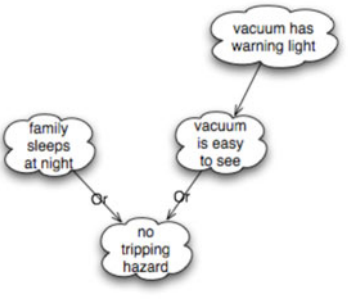


Fig. 2. Claim refinement model for *no tripping hazard* claim

to see is derived from the claim *vacuum has warning light*. The claims *family sleeps at night* and *vacuum is easy to see* together derive the bottom-level claim *no tripping hazard,* which is the claim underpinning the selection of the night cleaning strategy for the vacuum cleaner.

Falsity of any claim will propagate down the claim refinement model to the bottom-level claim. If the *family sleeps at night* claim is shown to be untrue or only partially true, the claim that there is *no tripping hazard* is upheld by the Or-ed claim *vacuum is easy to see*, provided the latter claim is sound. If confidence in all the claims in a claim refinement model was low, a validation scenario would have to be produced for every combination of true and false claims. However, some of the claims in the model may be axiomatic and the logic of claim derivations may give confidence in the bottom-level claims even where they are in part derived from non-axiomatic claims. To exploit this, a binary classification of claims, as axiomatic or conjectural, might be used:

Unbreakable claims are axiomatic and can be removed from consideration. *Vacuum has warning light* is such a claim.

Uncertain claims are conjectural in that it is considered possible that the claim could prove false at run-time. The claim that the *family sleeps at night*, and by implication would not encounter the vacuum cleaner, is conjectural because it cannot be assumed to be generally true. In contrast to the *vacuum is easy to see* claim, there is a significant possibility of this claim being proven false, since even a normally sound sleeper might awake and visit the bathroom while the vacuum cleaner is operating.

Using this classification a validation scenario should be developed for every bottom-level claim that resolves to *uncertain*, but need not be developed for one that resolves to *unbreakable*. Thus, for the robot vacuum cleaner, a validation scenario is needed to examine the consequences of the vacuum cleaner operating at night if the family was not asleep.

Unbreakable and *uncertain* represent the two extremes of the claim confidence spectrum. There may be claims that could be falsified but for which it is believed to be improbable that the system will encounter a situation where the claims are broken. We propose that such claims be classified as **Qualified.** However, a qualified claim should be re-classified *uncertain* if, despite the low probability of them being falsified, the consequences of them being so might be serious. Note that a claim with a high probability of falsification would already be classified *uncertain*, even if the consequences were considered minor.

The claim *vacuum is easy to see* might be classified as a qualified claim. Even a vacuum cleaner whose visibility is enhanced by a warning light may not be easily noticeable to people with visual impairment or people who are sleep-walking. However, sleep-walking is unusual and it is unlikely that robot vacuum cleaners would be recommended for people for whom they were obviously a hazard.

To propagate claim values (unbreakable, qualified, uncertain), the following rules apply:

- Where a claim is derived from a single claim (e.g. *vacuum is easy to see* is derived directly from *vacuum has warning light*), the derived claim inherits the value of the upper-level claim; the upper-level claim *make*s or *break*s the derived claim. Hence, *vacuum is easy to see* assumes the value unbreakable directly from *vacuum has warning light.*

- Where a claim is derived from two or more AND-ed claims, it inherits the value of the weakest of the claims – i.e. the claim least certain to be true. Hence if an unbreakable and an uncertain claim are AND-ed to derive a bottom-level claim, the bottom-level claim will be uncertain.
- Where a claim is derived form two or more OR-ed claims, it inherits the value of the strongest of the claims. Hence *no tripping hazard* assumes the value unbreakable from *vacuum is easy to see*, despite *family sleeps at night being* uncertain. In the example, this has the effect of validating selection of the *night-time cleaning* strategy in Fig 1.

The classification of claims cannot be easily formalized or automated. It requires human judgment by the analyst and stakeholders. However, if claims are classified thoughtfully and propagated through the claim refinement model, the result can be considered to be a form of threat modeling [12]. The exercise generates validation scenarios for the goal models but prunes back the set of all possible validation scenarios to include only claims that are judged plausible or high risk. In the next sections, we describe how claims can be used to extend LoREM, a previously-reported approach for modeling DAS requirements [6].

3 Modeling DASs with LoREM

LoREM offers a requirements modeling approach for DASs, following the principles set out in [13]. Here, to use Michael Jackson's terminology [14], the DAS represents the machine, while the environment or context in which it operates represents the world. The world is considered to comprise a set of discrete environmental states called *domains*. The machine is conceptualized as a set of distinct programs called *target systems*, each of which is designed to operate in a particular domain. In LoREM, each target system is explicitly specified, as are the domains and the conditions that trigger the DAS to adapt from one target system to another.

In LoREM, the set of domains is identified as part of the strategic dependency (SD) model. A different strategic rational (SR) model is then developed for each target system according to the principle of one target system for each domain, as demonstrated in the case study in the next section. Underpinning LoREM is an assumption that a DAS must satisfice a set of NFRs, but that the balance of satisficement and the consequent trade-offs differs according to domain. Hence, in the GridStix river flood warning system described below, *energy efficiency* and *fault tolerance* are in tension. Which is prioritized over the others depends on whether the river is (e.g.) quiescent or in flood. Using LoREM, these key NFRs are identified in the SD model as softgoals and each target system's SR model selects a solution strategy that optimizes the trade-offs in softgoal satisficements.

In [7] we augmented LoREM with claims. Claims are used in the target system SR models to annotate the contribution links between the tasks that represent the alternative solutions strategies and the softgoals, in exactly the same way as they were used for the vacuum cleaner model in Fig 1. The motivation for this was to make the requirements rationale (in terms of alternative selection) explicit and so aid tracing. Underpinning this rationale was the likelihood that, even with their self-adaptive capability, to have a usefully long life, a DAS would still be subject to off-line adaptive

and corrective maintenance over time. This would be necessary as understanding about its environment improved, as ways to improve its performance were identified and as its stakeholder's goals evolved.

As argued in the previous section, claims can also serve as markers for uncertainty, with conjectural or *uncertain* claims forming the focus of validation scenarios. A validation scenario captures a situation where combinations of uncertain claims are taken to be false. A claim refinement model is used to propagate the effects of the negated claims to other, derived claims. The validation scenario thus helps understand the impact on the validity of the solution strategies that those claims help select, either by static analysis or by testing.

At design time, the validation scenarios help answer what-if questions about the impact of the uncertainty that conjectural claims represent. Once the DAS is deployed, it may be possible to resolve many of the underlying uncertainties. If the validation scenarios have been applied correctly and complete system failure has not resulted from a broken claim, the resolved uncertainties may be put to good use.

A DAS monitors its environment and uses the monitored data to make adaptation decisions. In most DASs currently deployed, including those which LoREM was designed to support, a DAS can adapt in pre-determined ways to changes in its environment that were envisioned at design-time, albeit with elements of uncertainty. Where those envisioned environmental changes and the DAS's corresponding adaptations are based upon claims, the DAS may accumulate data that can be used to validate the claims. This is a form of requirements monitoring [5], the data from which may subsequently be used to inform off-line adaptive or corrective maintenance. Moreover, as we discuss in the conclusions section, it may even be possible to dynamically adopt an alternative solution strategy at run-time. At present, however, this is future work.

The next section presents a case study to illustrate the utility of claim reasoning in DAS goal models.

4 Case Study

To illustrate the use of claims for validation scenario identification in LoREM, we present a conceptually simple but real DAS and work through the model analysis for a single target system.

GridStix [15] is a system deployed on the River Ribble in North West England that performs flood monitoring and prediction. It is a sensor network with smart nodes capable of sensing the state of the river, processing the data and communicating it across the network. The hardware available includes sensors that can measure depth and flow rate, and wireless communication modules for the Wi-Fi and Bluetooth standards, all supplied with power by batteries and solar panels. The system software uses the GridKit middleware system [4], which provides the GridKit's self adaptive capabilities using component substitution.

The flow rate and river depth data is used by a *point prediction model* which predicts the likelihood of the river flooding using data from the local node and data cascaded from nodes further upstream. The more upstream nodes from which data is available, the more accurate is the prediction. GridStix acts as a lightweight Grid,

capable of distributing tasks. This is important because some tasks, such as execution of the point prediction models, can be parallelized and are best distributed among the resource-constrained nodes. However, distributing the processing comes at the cost of increased energy consumption and this is one of the factors that affect satisficement of the *energy efficiency* softgoal mentioned previously. Distribution may be effected by communicating between nodes using either the IEEE 802.11b (refered to as Wi-Fi in the rest of this paper) or Bluetooth communication standards. Bluetooth has lower power consumption, but shorter range. Bluetooth-based communication is hence thought to be less resilient to node failure than Wi-Fi-based communication. The choice of spanning tree algorithm can also affect resilience. A fewest-hop (FH) algorithm is considered better able to tolerate node failure than the shortest-path (SP) algorithm. However, data transmission via SP typically requires less power due to the smaller overall distance that the data must be transmitted.

The GridKit middleware can configure itself dynamically to support all the variations implied above; local node or distributed processing, Wi-Fi or Bluetooth communications and different network topologies, as well as others.

The environment in which GridStix operates is volatile, as the river Ribble drains a large upland area that is subject to high rainfall. The river is therefore liable to flooding with consequent risk to property and livestock among the communities sited on the flood plain. Stochastic models provide only an imperfect understanding of the river's behaviour. Moreover, events upstream, such as construction sites or vegetation changes, may alter its behaviour over time. There is therefore significant uncertainty associated with developing an autonomous sensor network for deployment on the river.

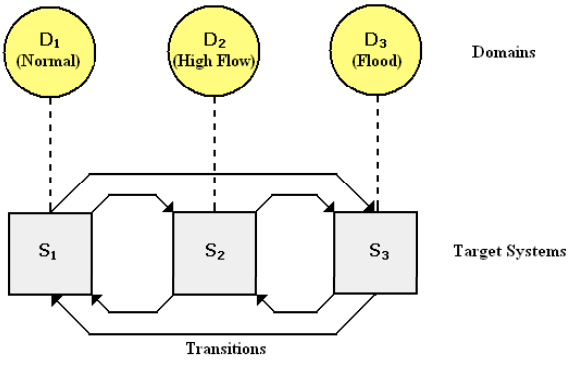


Fig. 3. Mapping Between GridStix Domains and Target Systems

The River Ribble, then is GridStix's environment. Hydrologists acting as domain experts partitioned the environment into three discrete domains: D_1 (Normal), D_2 (High Flow) and D_3 (Flood). The D_1 (Normal) domain is characterised by a quiescent river, with little imminent risk of flood or danger to local residents or the nodes themselves. The D_2 (High Flow) domain features a fast-flowing river that may presage an imminent flood event. The D_3 (Flood) domain occurs when the depth increases and

the river is about to flood. When this happens, GridStix itself is at risk of damage from node submersion and from water-borne debris.

GridStix is conceptualised as comprising three target systems, S_1, S_2 and S_3 tailored to domains D_1, D_2 and D_3 respectively, as shown in Fig 3.

The LoREM models for the system have previously been published in [6]. Here, we will focus on only one of the target systems to illustrate the use of claim reasoning. The SD model identified a single overall goal *Predict flooding*, and three softgoals; *Fault tolerance*, *Energy efficiency* and *Prediction accuracy*. The SR model for S_3 (Flood) is depicted in Fig 4.

Flood prediction in S_3 is operationalised by the task *Provide Point Prediction* which can be further decomposed into the (sub)goals: *Measure Depth*, *Calculate Flow Rate* and the task *Communicate Data*. The *Communicate Data* task depends on the *Transmit Data* and *Organize Network* subgoals being achieved. Satisfaction of the *Measure Depth* and *Calculate Flow Rate* goals produces *Depth* and *Flow Rate* resources respectively. These are used elsewhere in LoREM models which we do not consider here.

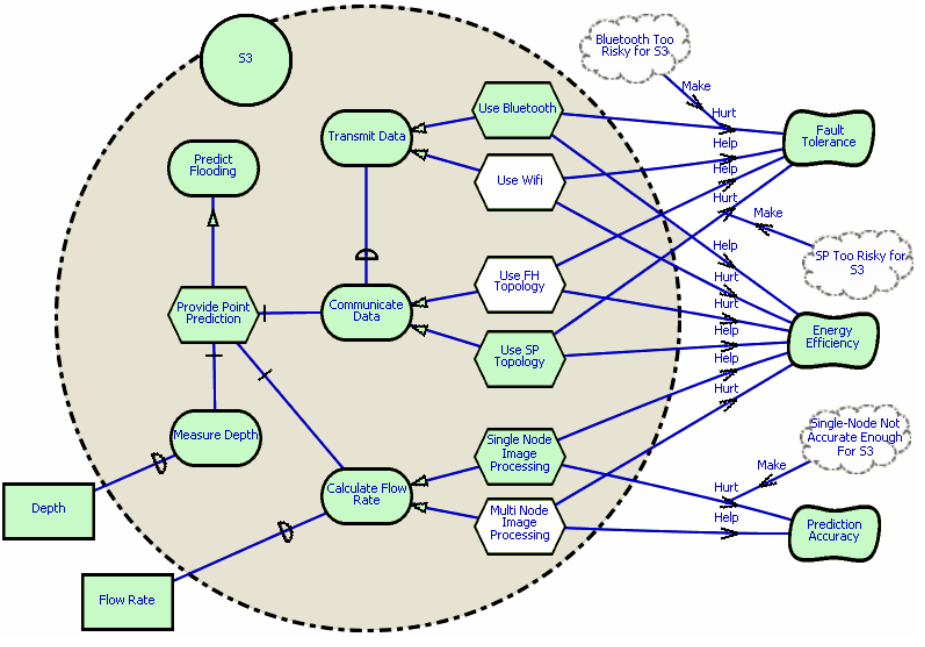


Fig. 4. GridStix Behaviour Model for S_3 (Flood)

The *Calculate Flow Rate*, *Organize Network* and *Transmit Data* goals all have several alternative satisfaction strategies, represented by tasks connected to the respective goals with means-end links. The three softgoals are depicted on the right of Fig 4, and the impact of selecting individual tasks on the satisficement of each of the softgoals is represented by contribution the links with values *help* or *hurt*.

The tasks represent alternative solution strategies and those selected to satisfy goals in S_3 are coloured white in Fig 4. Attached to some of the contribution links in Fig 4 are three claims that *make* or *break* the softgoal contributions. The claim refinement model for S_3 is depicted in Fig 5.

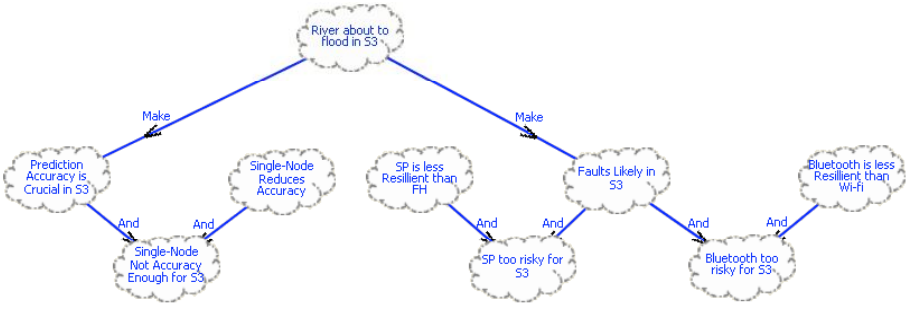


Fig. 5. GridStix Claim Refinement Model for S_3 (Flood)

The three claims shown in Fig 4 appear at the bottom of Fig 5, connected to the claims from which they are derived by contribution links. They show, for example, that Wi-Fi was selected over Bluetooth to satisfy the *Transmit Data* goal because Bluetooth was considered too risky in terms of Fault Tolerance. Examining Fig 5 allows the basis for this claim to be established: that Bluetooth is less resilient than Wi-Fi and that, given the river is about to flood in S_3, there is a significant risk of node failure. Bluetooth is considered less resilient than Wi-Fi because of its poorer range, which reduces the number of nodes an individual node may communicate with, so increasing the likelihood of a single node failure hampering communication throughout the network.

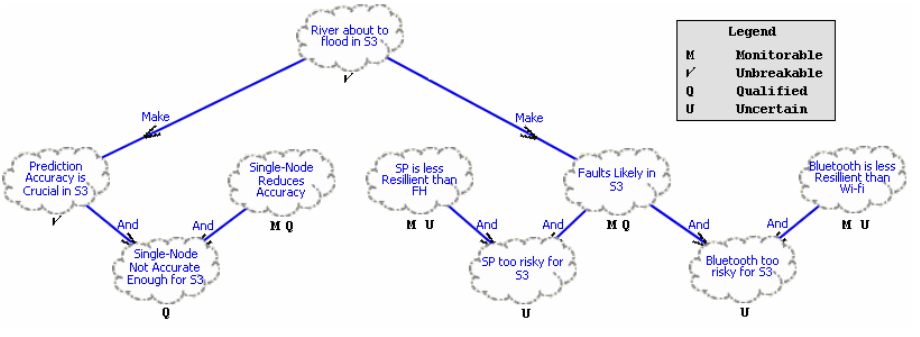


Fig. 6. GridStix Claim Refinement Model for S_3 (Flood) Annotated with Claim Classifications

There are three bottom-level claims in Figs 4 & 5, and we would thus need seven validation scenarios for every combination of claims. An analysis of the claims classified each according to the scheme presented in section 2. The results are shown in Fig 6.

Two claims were considered uncertain. The claims *SP is less resilient than FH* and *Bluetooth is less resilient than Wi-Fi* were both considered uncertain because the analyst was not certain whether the theoretically more resilient option in each case would actually prove demonstrably more resilient in the field. Note that the *Monitorable* tag (M) denotes a claim that could be directly monitored by the DAS at run-time. We return to claim monitoring in the conclusions.

Propagating the unbreakable, qualified, and uncertain values through the tree in Fig 6, shows that two of the three bottom-level claims, *SP too risky for S3* and *Bluetooth too risky for S3*, could be impacted by an uncertain claim and thus be uncertain themselves. The consequence of this analysis is that three validation scenarios are necessary. Recall that the purpose of the validation scenarios is to investigate the consequences of false claims, not to validate the claims themselves. The three identified validation scenarios contrast with the seven that would be necessary if all the claims were assumed to be falsifiable. The three scenarios are summarized in table 1.

Table 1. Validation scenarios for S3 uncertain claims

Scenario	*Single-Node[…] not accurate enough for S3*	*SP too risky for S3*	*Bluetooth too risky for S3*
1	True	True	False
2	True	False	True
3	True	False	False

Validation scenario 1, for example, would be designed to discover how GridStix behaved if the Bluetooth turned out to be no less fault-tolerant than Wi-Fi. Reasoning about the scenario concludes that GridStix's adaptations from Bluetooth to Wi-Fi would, at worst, simply waste energy (since Wi-Fi consumes more power than Bluetooth) should the *Bluetooth too risky for S3* claim indeed turn out to be false in practice. This is a relatively benign consequence but given the energy constraints on GridStix, it could be worth devoting effort to investigating the claims' validity, perhaps by simulation, or by monitoring the deployed system. In general, if a validation scenario reveals undesirable effects of a failed claim, there are two courses of action. As with the *Bluetooth too risky for S3* claim, additional effort might be assigned to understanding the claim's soundness. Alternatively, or in addition, an alternative solution strategy that circumvented the problem underlying the failed claim might be sought, using a new claim to record the rationale for its selection.

The same analysis was performed on the claim refinement models for the other two GridStix target systems: S_1 and S_2. The results of the analysis for all three are depicted in Table 2.

Table 2. GridStix Bottom-Level Claim Distribution by Target System

Target System	Unbreakable Claims	Qualified Claims	Uncertain Claims
S_1	2	1	2
S_2	1	0	2
S_3	0	1	2

The S_1 target system stands out as having the largest number of bottom-level claims (five), which reflects the complexity of the trade-offs in available solution strategies for this domain. Table 3 shows the numbers of validation scenarios that would be needed in each target system for all claims, qualified and uncertain claims, and uncertain claims only, respectively. This shows that using our strategy, the number of validation scenarios for S_1, even though it has five bottom-level claims, could be restricted to three for uncertain claims or seven if further assurance was felt necessary by including qualified claims.

Table 3. Emergent Behaviour Testing Scenarios for GridStix

Target System	All Scenarios	Qualified & Uncertain	Uncertain Scenarios
S_1	31	7	3
S_2	7	3	3
S_3	7	7	3
Total	45	17	9

Thus, devising validation scenarios for all combinations of uncertain claims would require nine scenarios, and for the qualified and the uncertain claims would require seventeen scenarios. Devising scenarios for all combinations of claims in the GridStix system would require forty-five validation scenarios.

The GridStix system, as presented here, is only modestly complex from a modelling perspective, with only three softgoals and three target systems. Many DASs will be significantly more complex. The effort involved in evaluating the consequences of poorly-informed softgoal trade-offs in a DAS increases rapidly with the number of potentially-conflicting NFRs involved. The technique described here has the potential to focus and limit this effort by the use of claims to explicitly record the rationale for solution strategy selection and by explicitly considering the soundness of those claims. An undesirable side-effect of using claims is that they add to the problem of complexity management in i* diagrams [21]. This is partially mitigated by the fact that we have designed claims to work with LoREM. Using LoREM, an SR model for a DAS is partitioned, with a separate SR model for each target system.

5 Related Work

There is currently much interest in software engineering for DASs [16]. Much of this work has been in the design of software architectures that enable flexible adaptations [2]. Much research in RE for self-adaptation has focused on run-time monitoring of requirements conformance [5, 17], which is crucial if a DAS is to detect when and how to adapt at run-time. More recently, attention has turned to requirements modeling for DASs, and a number of authors (e.g. [8, 18]) report on the use of goals for modeling requirements for DASs. Goal models are well suited to exploring the alternative solution strategies that are possible when the environment changes. Here, adaptation is seen as the means to maintain goal satisficement, while goal modeling notations such as KAOS [19] and i* [10] support reasoning about goals and softgoals.

A key challenge posed by DASs for RE is uncertainty about their environments and a number of modeling approaches for handling uncertainty have been proposed.

Letier and van Lamsweerde propose a formal means to reason about partial goal satisfaction. Cheng et al. [8] use KAOS's obstacle analysis to reason about uncertainty, utilizing a small set of mitigation strategies that include directly framing the uncertainty using the RELAX requirements language [20]. In this paper, we propose augmenting the i* models used by the LoREM approach to DAS modeling [6] with the *claim* construct adapted from the NFR framework [9]. We argue that by using claims as the rationale for selecting between alternative solution strategies, they can also serve as explicit markers for uncertainty where rationale is induced from assumed properties of the environment or the DAS's own behaviour.

There are two important differences between claims and the *belief* construct that is built in to i*. The first is that an i* belief represents a condition that an *actor* holds to be true. In our use of claims, the claim may also represent a condition that the analyst holds to be true. The second difference is that a belief attaches to a softgoal while a claim attaches to a softgoal's contribution link. Hence, a claim is able to provide the explicit rationale for selecting a particular solution strategy.

6 Conclusions

Claims attached to i* softgoal contribution links can be used to provide the rationales for selecting from several alternative solution strategies. Used this way, claims can be useful for tracing in DAS goal models [7]. Moreover, as we argue in this paper, claims may also be used as markers of uncertainty. The utility of claims may extend beyond DASs, but we focus on DASs because there is often significant uncertainty about a DAS's environment. Uncertainty may even extend to the DAS's own, *emergent* behaviour, if (e.g.) adaptation results in unexpected configurations of run-time-substitutable components.

Not all claims represent uncertainty, however. The confidence level in a claim will generally fall somewhere on a spectrum from axiomatic to pure conjecture. Conjectural claims represent uncertainty; assumptions that cannot be validated at design-time. Conjectural claims may therefore be falsified at run-time, possibly leading to a variety of undesirable effects. Accepting that such claims can't be easily validated at design-time, we should instead evaluate how the system will behave if a claim proves to be false by developing a *validation scenario*. A validation scenario subsumes a test case that may be developed for some combination of false claims, but also allows for static evaluation if the claims are hard to simulate in a test harness.

Validation scenarios may be costly to evaluate so the approach we advocate is designed to carefully select only those claims that have a significant probability of being false and those with a low probability of being false but whose falsification would be serious. To do this we advocate classifying claims as *unbreakable*, *qualified* or *uncertain*, and then propagating claim values through a claim refinement model.

As future work, we are developing the means to monitor claims at run-time, using the techniques of requirements monitoring [5]. Data collected about claim soundness may be used for subsequent off-line corrective maintenance. However, if the goal models can be treated as run-time entities where they can be consulted by the running system, the DAS may adapt by dynamically selecting alternative solutions when a claim is falsified. Such a system introduces new adaptive capabilities but also further

risk of undesired emergent behaviour so the identification of validation scenarios is acutely necessary. This run-time claim validation accounts for why we have chosen the name *unbreakable* rather than (e.g.) *axiomatic* for those claims at the opposite end of the spectrum from conjectural claims. Where claims can be validated and acted upon at run-time, we need to be able to monitor them. Not all claims are monitorable and the term unbreakable simply reflects that, at run-time, an unmonitorable claim can never appear to break from the perspective of the claim monitoring logic. Of course, such an unmonitorable claim may nevertheless be false and this may be exhibited externally as a failure.

References

1. Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: Software engineering for self adaptive systems. In: Dagstuhl Seminar Proceedings (2009)
2. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: FOSE 2007: 2007 Future of Software Engineering, pp. 259–268. IEEE Computer Society, Los Alamitos (2007)
3. McKinley, P., Sadjadi, S., Kasten, E., Cheng, B.: Composing adaptive software. Computer 37(7), 56–64 (2004)
4. Grace, P., Coulson, G., Blair, G., Mathy, L., Duce, D., Cooper, C., Yeung, W., Cai, W.: Gridkit: pluggable overlay networks for grid computing. In: Symposium on Distributed Objects and Applications (DOA), Cyprus (2004)
5. Fickas, S., Feather, M.: Requirements monitoring in dynamic environments. In: Second IEEE International Symposium on Requirements Engineering (RE 1995), York, UK (1995)
6. Goldsby, H., Sawyer, P., Bencomo, N., Cheng, B., Hughes, D.: Goal-Based modelling of Dynamically Adaptive System requirements. In: ECBS 2008: Proceedings of the 15th IEEE International Conference on Engineering of Computer-Based Systems, Belfast, UK (2008)
7. Welsh, K., Sawyer, P.: Requirements tracing to support change in Dynamically Adaptive Systems. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 59–73. Springer, Heidelberg (2009)
8. Cheng, H., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modelling approach to develop requirements of an adaptive system with environmental uncertainty. In: MODELS 2009: Procedings of IEEE 12th International Conference on Model Driven Engineering Languages and Systems, Colorado, USA (2009)
9. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Springer International Series in Software Engineering 5 (1999)
10. Yu, E.: Towards modeling and reasoning support for early-phase requirements engineering. In: RE 1997: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE 1997), Washington DC, USA (1997)
11. Department of Defence: DoD News Briefing - Secretary Rumsfeld and Gen. Myers, http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636
12. Schneier, B.: Attack Trees - Modeling security threats. Dr. Dobb's Journal (1999)
13. Berry, D., Cheng, B., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. In: 11th International Workshop on Requirements Engineering Foundation for Software Quality, REFSQ (2005)
14. Jackson, M.: Problem frames: analyzing and structuring software development problems. Addison-Wesley Longman, Amsterdam (2000)

15. Hughes, D., Greenwood, P., Coulson, G., Blair, G., Pappenberger, F., Smith, P., Beven, K.: Gridstix: Supporting Flood prediction using embedded hardware and next generation grid middleware. In: 4th International Workshop on Mobile Distributed Computing (MDC 2006), Niagara Falls, USA (2006)
16. Cheng, B., et al.: Software engineering for self-adaptive systems: A research road map. In: Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. Dagstuhl Seminar Proceedings, vol. 08031 (2008)
17. Robinson, W.: A requirements monitoring framework for enterprise systems. Requirements Engineering 11(1), 17–41 (2006)
18. Lapouchnian, A., Liaskos, S., Mylopoulos, J., Yu, Y.: Towards requirements-driven autonomic systems design. In: DEAS 2005: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software (DEAS), St. Louis, MO, USA (2005)
19. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley & Sons, Chichester (2009)
20. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B., Bruel, J.-M.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In: Proc. 17th IEEE International Conference on Requirements Engineering (RE 2009), Atlanta, Georgia (August 2009)
21. Moody, D., Heymans, P., Matulevicius, R.: Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of the i* Visual Notation. In: Proc. 17th IEEE International Conference on Requirements Engineering (RE 2009), Atlanta, Georgia (August 2009)

Use of Personal Values in Requirements Engineering – A Research Preview

Rumyana Proynova[1], Barbara Paech[1], Andreas Wicht[2], and Thomas Wetter[2]

[1] University of Heidelberg, Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{proynova,paech}@informatik.uni-heidelberg.de
[2] University of Heidelberg, Im Neuenheimer Feld 305, 69120 Heidelberg, Germany
andreas.wicht@uni-heidelberg.de, thomas.wetter@urz.uni-hd.de

Abstract. [Context and motivation] During requirements engineering the stakeholder view is typically captured by scenario- and goal models focusing on tasks and goals to be achieved with the software. We believe that it is worthwhile to study more general personal values and attitudes of stakeholders and to relate them to software requirements. [Question/problem] The main questions of such an approach are: what values can be expected from stakeholders, how can they be elicited and what can be learned from them for requirements. [Principal ideas/results] The purpose of our research is to provide a value elicitation technique to be combined with existing requirements elicitation techniques in order to infer additional ideas for and constraints on software requirements. [Contribution] In this paper we give a preview on our approach to developing such an elicitation technique. We start with an introduction to the theory of personal values. Then we describe our envisioned approach how values can be used in the requirements engineering process.

Keywords: requirements elicitation, values.

1 Introduction

It is widely known from practice that the requirements engineering (RE) process is heavily influenced by soft issues such as politics or personal values of stakeholders, but there is very little guidance on how to deal with these issues [1] [2]. Goals models such as i* [3] include goals, softgoals and actor dependencies, but give only little guidance on how to elicit these intentional elements, and don't call for a deeper analysis of these elements. Therefore they often capture only quite apparent economic or operational goals. Scenario-oriented approaches typically incorporate guidelines from human-computer interaction to focus on user tasks or use cases and to include early prototyping [4], but they do not capture information about user motivation.

We believe it is important to reveal the fundamental issues behind goals and task performance and to incorporate them into current RE approaches. Therefore, we propose to study personal values and their relationship to software requirements. We chose personal values because they are an important motivation factor, which remains stable independent of context [5]. We expect that the effect of personal values on requirements will be especially pronounced in the health care domain where effective patient treatment is the focus of the work of physicians and nurses.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 17–22, 2010.

We have recently launched an interdisciplinary research project with experts from software engineering and medical informatics whose goal is to examine how personal values influence software requirements in health care. Our long-term goal is to construct a practical approach for requirements elicitation that incorporates our insights into personal values. This paper presents the direction of our research. It is structured as follows: in section 2, we describe the psychological theory we base our ideas on. We explain briefly what values and attitudes are, how they affect human behavior, and what methods are commonly used for their elicitation. In section 3, we provide an overview of the approach we intend to develop in our project. Section 4 outlines other related work. We conclude with an outlook.

2 Personal Values

Motivation has always been an important research topic in psychology: what makes an individual behave in a certain way? The widely accepted human value theory introduces the concept of *personal values* as a major behaviour determinant for the individual. Most contemporary publications on value theory build on the work of social psychologist Shalom Schwartz [6], who validated his theory using extensive empirical studies. Note that there are behaviour influences other than personal values, such as economic values or emotions. In our research we focus on personal values, subsequently called "values".

Schwartz defines values as "desirable, trans-situational goals, varying in importance, that serve as guiding principles in people's lives."[7]. The connotation of "goal" in this definition is slightly different than the one typically used in RE literature: personal values like social recognition and free choice are seldom modelled among stakeholders' goals. While goal-oriented RE concentrates on stakeholder goals limited to a single purpose, values function on a much higher level. They are deeply ingrained in culture and the individuals acquire them during the socialization process. Individuals generally behave in a way which helps them achieve these values. Exceptions from this rule arise in situations where other behavioural determinants are predominant, such as biological needs or ideological prescriptions. In early studies, Schwartz discovered ten common values exhibited to a different extent by all participants. Table 1 lists these values and short explanations for each. Since then, extensive research has shown that these ten values occur independently of race, nationality, social and cultural background. This is an important conclusion of value theory: different populations don't strive for fundamentally different values, but there is a set of values common to all of us.

Despite sharing the same values, different individuals act differently in similar situations. The reason is that they place different importance on each value. In a situation where there is a conflict between values (e.g. donating money to a charity aids benevolence and universalism, but spending the same money on one's hobby helps achieve hedonism), an individual would choose the option consistent with the values he or she deems more important. So while everyone believes in the desirability of the same values, each individual has a personal ranking of the values.

Table 1. The ten values found by Schwartz (based on [7])

Achievement: Personal success through demonstrating competence according to social standards.
Benevolence: Preservation and enhancement of the welfare of people with whom one is in frequent personal contact.
Conformity: Restriction of actions, inclinations and impulses likely to accept or harm others and violate social norms or standards.
Hedonism: Pleasure and sensuous gratification to oneself.
Power: Social status and prestige, control and dominance over people and resources.
Security: Safety, harmony and stability of society, of relationship, and of self.
Self-direction: Independent thought and action-choosing, creating, exploring.
Stimulation: Excitement, novelty and challenge in life.
Tradition: Respect, commitment and acceptance of the customs and ideas that traditional culture or religion provide the self.
Universalism: Understanding, appreciation, tolerance and protection for the welfare of all people and for nature.

Values are important in RE because of the way they shape the individual's interactions with software. They are a criterion for the evaluation: the desirability of a behaviour option increases monotonically with the degree in which performing it helps the individual achieve the (high-ranked) values. The evaluation process may be deliberate or subconscious. But regardless of the awareness of the actual process, the individual is usually aware of its outcome. He or she can articulate it as a statement of the kind "I like X" or "I don't like Y". Such judgments of behaviour options (and also judgments of any other entities) are known in psychology as *attitudes*.

An attitude is defined as "a psychological tendency that is expressed by evaluating a particular entity with some degree of favour or disfavour" [8]. Unlike the universally applicable values, an attitude always is about some target entity. It always implies an evaluation, which may use one or more dimensions (like/dislike, good/bad), but invariably results in an aggregated assessment, positive or negative.

Attitudes are formed through a judgment process. Since values are important criteria for our judgment, attitudes are at least partly based on values. Thus, when information on values isn't available, information on attitudes can be used as an indicator for the ranking of values [9]. This has important implications for empirical research. As attitudes are much more salient than values, their self-reporting proves easier than self-reporting on values. Thus, researchers interested in values can apply instruments based on attitudes, which are more reliable, and then use the results to draw conclusions about the subjects' values (see e.g. [10]). But the correlation can also be used in the opposite direction: once the connection between value rankings and attitudes toward specific targets is supported by empirical evidence, knowledge of an individual's value ranking can be used (given some conditions described in [11]) to predict his or her attitude toward these targets.

3 Relating Personal Values and Requirements

Research on organizational psychology has shown that the way a person works strongly depends on his or her general motivation factors [12]. Therefore, important motivation factors such as values are likely to have impact on the user satisfaction with a specific software product. Our goal is a value elicitation approach which can be used parallel to existing requirements elicitation methods to discover useful information which usually isn't explicitly stated. As depicted in Figure 1, our approach will provide a method to elicit values, a method to infer attitudes towards tasks from values, and a method to elicit requirements details from these attitudes.

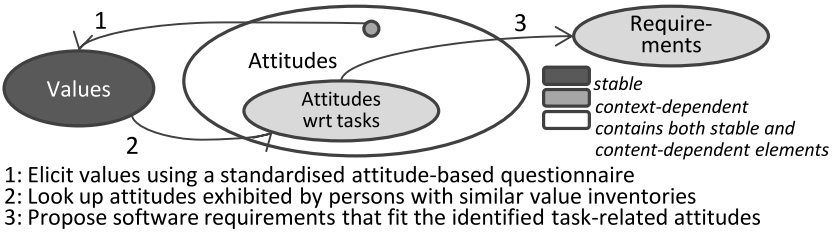


Fig. 1. Our proposed value-based elicitation approach

The first step of our approach is the elicitation of values. Existing instruments for value elicitation might appear too intrusive in requirements engineering practice, because in this situation users might be reluctant to answer direct questions about their personality. As stated in section 2, some attitudes strongly correlate with values. We plan to develop as part of the first method a new attitude-based questionnaire which gives us information about a user's values. We want to explore how acceptance of the original Schwartz questionnaire compares to our instrument.

When the values are known, the requirements analyst can use our second method to predict the user's attitudes towards different tasks. The seemingly superfluous round-trip from attitudes to values and from that to other attitudes is caused by the fact that directly questioning a user about his or her attitudes towards the hundreds of software supported tasks involves too much effort. Moreover, the attitudes towards tasks are situation dependent and likely to change as soon as the context changes. On the other hand, values are an integral part of the user's personality and unlikely to change [13]. They allow us to predict how the user's attitudes will change after a context change.

We plan to identify for our second method correlations between values and attitudes towards tasks at least for the medical domain. But even when a catalogue of empirically founded statements about value-attitude correlations isn't available, a requirements analyst with basic knowledge of value theory can use the information on the values to reason about expected attitudes. For example, if the value elicitation shows that the user is primarily motivated by the value stimulation, which is mediated through novelty, then it is reasonable to assume that whenever faced with a task like "record patient temperature", the user would prefer to input the data using a trendy electronic device instead of scribbling on paper.

In the third step, the analyst uses the information about attitudes and our third method to enrich the existing requirements. Our approach does not include the identification of tasks and goals; they have to be elicited using classical methods. But knowing the users' attitude towards the tasks allows deeper insight into the requirements. It can be especially useful for uncovering new requirements which weren't verbalised by the user: if a physician has a negative attitude towards tasks involving typing, possibly associated with a value of achievement or power, we can expect him or her to take notes on patient symptoms on paper and delegate the data input to a nurse. This means that he or she needs a system which gives nurses writing access to patient records. Of course, such an inferred requirement cannot be simply included in the specification without discussing it with the relevant stakeholders. But the merit of the value-based approach in this example is that it has revealed the existence of an issue which could be easily overlooked in a traditional process oriented task description.

4 Related Work

By wording, value-based software engineering seems related to our research [14]. However, so far it typically focuses on economic value, not on personal value. Only recently have personal values been addressed in RE [2]. This publication also discusses motivation, but it uses a much broader definition of the term "value", namely any concept which influences human behaviour is a value. Furthermore, it considers other soft issues such as emotions or motivations. Our understanding of "value" is roughly equivalent to their notion of a "motivation". So while our research has a similar focus, that publication remains on a much more general level.

Psychology provides plenty of literature on personal values. We name some of the main sources in this publication. Psychology also offers many studies on the link between work patterns and personal behavioural determinants like values, beliefs etc. Some of these studies focus on health care professionals, such as [12], [15]. They provide valuable insights in the motivation of clinicians, but don't link them to their software use or software requirements. Another type of studies concentrates on professionals' attitudes towards computers in general [16], but we aren't aware of any results which try to establish a link between attitudes and software requirements.

5 Conclusion and Outlook

We have presented the theory of personal values and our approach on how to support the RE process. We aim to achieve two results. First, we want to develop a theoretical description of an approach for eliciting new requirements based on values, as described in Section 3. Second, we want to provide knowledge needed for applying our approach in practice, such as a catalogue with some common relations between values and requirements.

We are currently preparing first empirical studies. In parallel we expand our literature study to include information systems literature on technology acceptance such as [17]. We will use results from our first interview stage to establish hypotheses about relations between specific values and requirements. In subsequent stages, we will try to verify our hypotheses by collecting data from new study participants.

References

1. Robertson, S., Robertson, J.: Mastering the Requirements Process, 2nd edn. Addison-Wesley, Harlow (2006)
2. Thew, S., Sutcliffe, A.: Investigating the Role of 'Soft Issues' in the RE Process. In: Proceedings of the 2008 16th IEEE International Requirements Engineering Conference. IEEE Computer Society, Los Alamitos (2008)
3. Yu, E.S.K.: From E-R to 'A-R' - Modeling strategic actor relationships for business process reengineering. International Journal of Cooperative Information System 4, 125–144 (1995)
4. Lauesen, S.: Software Requirements: Styles and Techniques. Pearson Education, London (2001)
5. Wetter, T., Paech, B.: What if "business process" is the wrong metaphor? Exploring the potential of Value Based Requirements Engineering for clinical software. In: Accepted at MedInfo 2010, CapeTown (2010)
6. Schwartz, S., Bilsky, W.: Toward a theory of the universal content and structure of values: Extensions and cross-cultural replications. Journal of Personality and Social Psychology 58, 878–891 (1990)
7. Schwartz, S., Melech, G., Lehmann, A., Burgess, S., Harris, M., Owens, V.: Extending the Cross-Cultural Validity of the Theory of Basic Human Values with a Different Method of Measurement. Journal of Cross-Cultural Psychology 32, 519–542 (2001)
8. Eagly, A., Chaiken, S.: The psychology of attitudes. Harcourt Brace Jovanovich College Publishers Fort Worth, TX (1993)
9. Bohner, G., Schwarz, N.: Attitudes, persuasion, and behavior. In: Tesser, A. (ed.) Blackwell handbook of social psychology: Intrapersonal processes, vol. 3, pp. 413–435. Blackwell, Malden (2001)
10. Inglehart, R.: Modernization and Postmodernization: Cultural, Economic, and Political Change in 43 Societies. Princeton University Press, Princeton (1997)
11. Ajzen, I., Fishbein, M.: The influence of attitudes on behavior. In: Albarracin, D., Johnson, B., Zanna, M. (eds.) The handbook of attitudes, vol. 173, p. 221. Lawrence Erlbaum, Mahwah (2005)
12. Larsson, J., Holmstrom, I., Rosenqvist, U.: Professional artist, good Samaritan, servant and co-ordinator: four ways of understanding the anaesthetist's work. Acta Anaesthesiol Scand 47, 787–793 (2003)
13. Rokeach, M.: The nature of human values. Jossey-Bass, San Francisco (1973)
14. Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grunbacher, P.: Value-based software engineering. Springer, New York (2006)
15. Timmons, S., Tanner, J.: Operating theatre nurses: Emotional labour and the hostess role. International Journal of Nursing Practice 11, 85–91 (2005)
16. van Braak, J.P., Goeman, K.: Differences between general computer attitudes and perceived computer attributes: development and validation of a scale. Psychological reports 92 (2003)
17. Venkatesh, V., Bala, H.: Technology acceptance model 3 and a research agenda on interventions. Decision Sciences 39, 273 (2008)

Requirements and Systems Architecture Interaction in a Prototypical Project: Emerging Results

Remo Ferrari[1], Oliver Sudmann[2], Christian Henke[2], Jens Geisler[2], Wilhelm Schafer[2], and Nazim H. Madhavji[1]

[1] University of Western Ontario, London, Canada
{rnferrar,madhavji}@csd.uwo.ca
[2] University of Paderborn, Paderborn, Germany
{oliversu,henke,jgeisler,wilhelm}@upb.de

Abstract. [Context and Motivation] Subsequent to an exploratory laboratory study on the effects of Software Architecture (SA) on Requirements Engineering (RE), in this paper, we present preliminary results of an extension of this initial study by conducting a case study on a large-scale prototypical rail project. **[Question/Problem]** Specifically, we ask "What is the role of an SA on Requirements decision-making?". **[Principal Ideas/Results]** Specific types of architectural effects on requirements decisions are identified. The impact of the affected requirements decisions on downstream processes and the product itself is also characterized. **[Contribution]** The understanding gained from this study has implications on such areas as: project planning, risk, RE, and others.

1 Introduction

A recent, laboratory, study of ours [3] investigated the issue of the impact of an existing software architecture (SA) in Requirements Engineering (RE), where we identified four types of RE-SA interaction effects along with their quantitative profile: (i) constraint (25%), if the existing SA makes a requirement solution approach less (or in-) feasible; (ii) enabler (30%), if the existing SA makes a solution approach (more) feasible because of the current architectural configuration; (iii) influence (6%), if the architectural effect altered a requirements decision without affecting the feasibility of its solution approaches; and (iv) no effect (39%), if the architecture has no known effect on a requirements solution.

In this paper, we present emerging results of a replicated case study on a large-scale prototypical rail project (RailCab) being conducted in Germany. While we continue the main investigation of [3], here we also investigate the impact of the affected requirements decisions on downstream development processes and the resultant system. The case study, which is still ongoing, involves the investigation of the history of requirements and architecting decisions in five major components of RailCab (e.g., drive and brake, suspension/tilt and active guidance). For the emerging results reported in this paper, data for one of these components (Energy Management) was collected from project documents and extensive interviews with the RailCab developers and planners. The results of this study have implications for: project

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 23–29, 2010.

planning and risk management, tighter RE-SA integration, RE technology, value-based traceability, requirements elicitation, and future empirical research in RE.

In the next section we describe the case study; in Section 3 we present the preliminary results; Section 4 discusses example implications; in Section 5 we summarize related work; and lastly, Section 6 concludes the paper.

2 The Case Study

In this section, we raise the research questions; overview the RailCab project; and describe the participants, and empirical procedures.

Research Questions: We ask two pertinent questions,

Q1: *What is the impact of an existing system's architecture on RE decision-making?*
This question replicates the investigation in [3] on the impact the presence of an architecture has on decision-making in RE. Basically, a RE decision denotes a chosen subset of high-level requirements (or solution strategies) amongst a set of alternatives in order to achieve a goal. For example, deciding to provide a web-based self-help service to clients (as opposed to phone-in service or personal contact service) in order to cut down operational costs. It is through the choice of such high-level business strategies that detailed requirements are then elicited and established. When individual requirements are elicited, also considered then are related attributes such as assumptions and issues (such as cost implications, constraints, actions, etc.) [5]. Thus, an individual requirement is only "indirectly" related to a RE decision through identified strategies [3]. This notion of requirements decisions is operationalised through the decision meta-model designed and validated in [3].

A point to note is that all RE decisions are not of same importance; some decisions may affect the system or development process more significantly than others. Thus, the primary criterion for establishing whether a RE decision should be included in the set for data analysis is whether the stakeholders themselves feel that a particular decision is important for them. In this study, the stakeholders are the participants of the study and the decisions were extracted from their interviews and project documents, and later validated by them for completion and accuracy.

The research question Q1 is investigated by collecting and analyzing the data from two constructs: the requirements decisions and their RE-SA interaction type (e.g., constrained, enabled, influenced or neutral – see Section 1).

Furthermore, in this paper we extend the analysis from [3] by probing deeper into the identified affected decisions, resulting in the following new question:

Q2: *What is the impact of the affected requirements decisions on the resultant system and downstream development activities?*
The second question focuses on the affected decisions identified through the investigation of Q1. Note that question Q2 is examined from two angles: product and process. For each affected decision, we interviewed the project (RailCab) staff to determine the impact of RE decisions on the system and activities outside of the requirements elicitation process, such as implementation and testing processes, system reliability, safety, and others.

Study Context: The RailCab project has been in development for approximately ten years at the University of Paderborn in Germany. The goal of the project is to develop a comprehensive concept for a future railway system. RailCab is considered a "mechatronic" system, i.e., it requires the interdisciplinary expertise in the areas of mechanical, electrical and software engineering fields. The key feature of RailCab is that it is an autonomous, self-optimizing system, and thus does not require any human operator. The RailCab consists of five major components: Drive and Brake, Energy Management, Active Guidance, Tilt and Suspension, and Motor and Track Design. For this short paper, we examine the Energy Management component only. The primary purpose of this component is to ensure that each of the RailCab subsystem's energy demands are fulfilled. Additionally, the component is responsible for recharging the energy sources as the train operates. Other features include heat and voltage monitoring of battery arrangements for safety purposes, using batteries as main power supply for driving if track energy is not available, and adjusting energy levels at runtime based on differing priority levels of subsystems requesting energy. The component involves a mix of hardware and software elements; the software is executed on an on-board computer and controls the various functions of the component listed above[1].

Participants, Data Collection, Data Analysis: In this study, eight senior-level developers and researchers (with over five years of experience and domain expertise) were extensively interviewed over a span of four months on a bi-weekly basis for approximately one hour each interview session. Each developer is primarily responsible for his/her own major component. Additionally, they provided project documents and validated emergent findings.

There are numerous qualitative-based data sources in this project: minutes, theses and reports, research papers, presentation slides, design documents, prototypes, other project documents. Other primary data source is the RailCab developers interviews. Over ten hours of audio recordings resulted in over 80 transcribed pages of text. The interviews focused on: (i) domain understanding, (ii) extracting the requirements decisions and high-level architecture for the Energy Management component, and (iii) determining the RE-SA interaction.

Qualitative coding [6] was used to analyse the project documents and interview data. For example, if a segment of text is describing a current requirement for the "Active Guidance" module (one of the components of the RailCab system) and how the requirement is affected by a previous architectural decision made for the "Energy Management" module then it will be tagged as such. The coded text can then be counted to create frequency figures of the various categories (i.e., requirements decisions) which form the basis of the study results.

3 Emerging Results

3.1 Architectural Impact on RE Decision-Making (Q1)

In the Energy Management component, a total of 30 requirements decisions were extracted from the project documents and interviews with the RailCab staff. A significant

[1] Because of space constraints, we are not able to provide comprehensive information regarding the technical SA details of the RailCab. For more information, the readers are referred to: http://www.sfb614.de/en/sfb614/subprojects/project-area-d/subproject-d2/

portion of these decisions was affected in some way by the evolving architecture. Here, we describe the characteristics of these RE-SA interactions.

Referring to Table 1, overall, 13 out of the 30 decisions (43%) were affected by previous architectural decisions. Conversely, 17 decisions (57%) were not affected. Out of the 13 affected decisions, 8 were of the type constrained (27%) and 6 were of type enabled (20%), almost an even split between these two types. These figures are similar to what we observed in our previous study [4], where 30% of the effect types were found to be of type enabled and 25% were of type constrained.

Table 1. RE-SA interaction effect profile

	# of Reqt's Decisions	Constrained	Enabled	Influenced
Affected	13 (43%)	8 (27%)	6 (20%)	0 (0%)
Not affected	17 (57%)			

The effect type influenced was 6% in our previous study [3] to no observations in the current component study. No new types of RE-SA interaction effects were observed in the component study. The overall affected:not_affected results (43%:57%) are similar to that in our earlier study in the banking domain (59%:41%). We now probe into the characteristics of the different types of affected decisions.

Constrained. The 8 out of 30 (27%) constrained decisions can be considered substantive. Of these, two decisions were core or essential for the operation of the RailCab. These were the availability of the driving of the shuttle vehicle even in case of track power failure, and that the hydraulic unit receives its own energy converter and supply. Four of the eight constrained decisions are what can be classified as "consequential" decisions, i.e., decisions that emerged as a consequence of other decisions made in the same subsystem, or were triggered by feedback from other implementation-based development activities. The remaining two decisions were triggered by design oversights made previously in other components that were not discovered until the implementation and testing phases of development.

Enabled. The 6 out of 30 (20%) *enabled* decisions can also be considered substantive. Of these, 4 decisions led to core requirements for the energy subsystem. One decision was consequential and emerged during more detailed construction and implementation phases. Finally, one decision was the mixed enabled/constrained decision and its source was both as a core feature of the RailCab, and was also a fix to a previous design oversight from the motor and track topology design.

Neutral. It is also significant that 17 out of 30 (57%) decisions were not affected by previous architectural decisions. Basically, these decisions were made during the early phases of planning, which spanned approximately 2-3 years, and remained stable for the entire duration of the development process. Furthermore, these decisions and their subsequent solutions were largely dictated by the system domain and did not offer many alternative solution strategies.

3.2 Impact of Affected Decisions on Processes and System (Q2)

In every case of a constrained decision, the result was increased construction (i.e., hardware assembly and software coding) time and effort due to the constrained decision. The second highest development activity that was severely affected by constrained decisions was testing, which was observed in 7 out of the 8 cases. This involved the creation of new test cases as well as testing procedures. The third highest development activity was systems architecting, which was affected in terms of effort spent in 5 out of the 8 cases due to a constrained decision. Other activities (e.g., requirements prioritization, costing, and elicitation) were also observed but in only 1 or 2 cases.

For the enabled cases, the impact on other activities is difficult to discern since there are no counter-cases to compare; the benefit could only really be observed during the RE decision making time but later in the implementation, design and testing it was not reported any different than the not affected decisions.

The impact of the constrained decisions on system properties (i.e., non-functional attributes) – in the context of this single component study -- was less noticeable. In 5 of the 8 cases, the developers reported a slight degradation of system quality. The attributes mostly affected were the physical space of the shuttle (in 3 cases) and the software modifiability of the system (in 2 cases); the only other attribute affected (in 1 case) was the energy efficiency. However, in all of these cases, the system properties did not deviate significantly from what was originally intended or desired. As with the previous subsection, it was difficult to discern any positive benefit in the enabled cases because they followed a similar implementation path as the not affected cases.

These results seem to fit the characteristics of a "prototypical" project where, in a constrained decision, developers could not simply upgrade or replace hardware components because of the cost involved; instead, they had to spend time and effort in finding alternative solutions that still provided near-desired levels of system quality. In a production environment, the reaction to constrained decisions may very well different, depending on budgetary and other factors.

4 Implications

There are a number of implications of the findings. We discuss one example:
Tighter SA-RE integration across different subsystems: With almost 50% of the RE decisions being affected by an architecture (see Table 1), and many (50%) of the affected decisions originating outside of the energy management component, it is strongly encouraged that the SA and RE processes be more tightly integrated [4] to provide insight on the technical feasibility of the elicited requirements in terms of constraints and enablers from a non-local sub-system.

In RailCab, RE and SA decisions are predominantly made synonymously within a single subsystem and no distinction is made between SA and RE roles. For example, the early decisions for the motor and track topology subsystems led to constraints in the energy management system which the planners knew about but deferred until later. Requirements and Design were highly intertwined in the motor and track subsystem, yet during this early planning phase the focus was almost entirely on the

motor and track subsystem; high-level requirements were elicited for the energy subsystem but no detailed RE or SA work was done at that time. After the motor and track design phases were near completion, the energy subsystem's detailed RE and SA phases commenced. However, it was then determined that previously known constraints would be more difficult to plan and implement because of tradeoffs introduced in design decisions from the motor and track subsystem. Thus, one lesson learnt from this is that during the design phase, corresponding detailed RE and SA work should also have been carried out in the energy subsystem to handle alignment issues.

5 Related Work

From [3], we summarize below related work on the role of an SA in RE. In 1994, Jackson [2] gave four key reasons why RE and SA are best treated as interweaving processes. In 1995, El-Emam and Madhavji [1] found four factors for RE success in information systems that deal with architecture and/or the system (one of which is relevant for this study): the adequacy of diagnosis of the existing system (which includes SA). In 2001, Nuseibeh [4] described the "twin-peaks" model, which captures the iterative relationship between RE and SA. An important aspect of this model is that SA can, and should, feed back into the RE process.

6 Conclusion

We describe the impact an existing Systems Architecture has on requirements decisions, determined through a case study on a rail project (RailCab), and is an extension of an initial exploratory study [3] that was conducted in a "laboratory" setting. The case study involved the analysis of approximately 10 years worth of project documents and extensive interviews with RailCab staff – with a focus on one of the six major RailCab system components (Energy Management). In a nutshell, we found 30 requirements decisions where:

- 13 (43%) were affected by a previous architectural decision.
- 8 of these 13 decisions were *constrained* by the existing architecture.
- 6 of these decisions were *enabled* by the existing architecture.

Furthermore, from the identified affected requirements decisions, we qualitatively determined their impact on other development activities and properties of the resultant system. Despite being emergent findings, this early evidence suggests that existing architecture in the Requirements Engineering process does have a serious impact on requirements decisions.

Acknowledgements. This work was, in part, supported by Natural Science and Engineering Research Council (NSERC) of Canada. This contribution was also developed and published in the course of the Collaborative Research Center 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" funded by the German Research Foundation (DFG) under grant number SFB 614.

References

1. El Emam, K., Madhavji, N.H.: Measuring the Success of RE Processes. In: Proc. of the 2nd IEEE Int. Symp. on RE, York, England, March 1995, pp. 204–211 (1995)
2. Jackson, M.: The Role of Architecture in RE. In: Proc. of 1st Int. Conf. on RE, p. 241 (1994)
3. Miller, J., Ferrari, R., Madhavji, N.H.: Architectural Effects on Requirements Decisions: An Exploratory Study. In: 7th Working IEEE/IFIP Conf. on SA, Vancouver, Canada, pp. 231–240 (2008)
4. Nuseibeh, B.: Weaving Together Requirements and Architectures. IEEE Comp. 34(3), 115 (2001)
5. Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering 2(1), 58–93 (2001)
6. Runeson, P., Host, M.: Guidelines for conducting and reporting case study research in software engineering. Journal of Emp. Soft. Eng. 14(2), 131–164 (2009)

Videos vs. Use Cases: Can Videos Capture More Requirements under Time Pressure?

Olesia Brill, Kurt Schneider, and Eric Knauss

Software Engineering Group, Leibniz Universität Hannover,
Welfengarten 1, 30167 Hannover, Germany
{olesia.brill,kurt.schneider,eric.knauss}@inf.uni-hannover.de

Abstract. [Context and motivation] Many customers and stakeholders of real-world embedded systems are difficult to reach with traditional requirements elicitation and validation techniques. Traditional requirements engineering methods do not deliver concrete results for validation fast enough; stakeholders get impatient or misunderstand abstract requirements. **[Question/problem]** The problem is to achieve a mutual understanding between customers and the requirements engineer quickly and easily, and to get stakeholders involved actively. **[Principal ideas/results]** We propose to use ad-hoc videos as a concrete representation of early requirements. Videos have been used before in requirements engineering: Sophisticated videos were created at high effort. We show, however, that even low-effort ad-hoc videos can work comparably or better than use cases for avoiding misunderstandings in the early phases of a project. **[Contribution]** We replicated and refined an experiment designed using the Goal-Question-Metric paradigm to compare videos with use cases as a widely used textual representation of requirements. During the experiment, even inexperienced subjects were able to create useful videos in only half an hour. Videos helped to clarify more requirements than use cases did under the same conditions (i.e. time pressure).

Keywords: Empirical Software Engineering, Video-based Requirements Engineering.

1 Introduction and Context

Recently, the use of videos in Requirements Engineering (RE) has been proposed [1, 2, 3, 12]. Initially, videos were mainly considered dynamic prototypes that show how the system under construction would finally be used. Videos are valuable in this area, because they provide stakeholders with a clear vision of what the system should do. According to Anton et al. this allows stakeholders to give better, i.e. more and less volatile, feedback [1].

In more recent research, videos were proposed for actually documenting requirements as well [2, 3]. These works indicate that it is possible to use videos as requirements documents. This is a nice option, because it allows documenting requirements in a way most useful for stakeholder interaction [2, 3]. We assume that these effects

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 30–44, 2010.

are most valuable during elicitation and validation activities. During these activities it is often hard to reach customers and stakeholders. In addition, customers and stakeholders have limited time, get impatient, and even misunderstand abstract requirements. In order to validate requirements, a concrete representation of requirements (e.g. a prototype) is needed. Videos as a means of documenting requirements promise to support stakeholder interaction in a more efficient way, because they are more concrete and easier to understand by stakeholders. Yet, in literature it remains unclear if there are any advantages over textual requirements documents. Empirical insights about costs and benefits of videos are needed.

In this work we investigate whether videos can replace textual requirements representations. We give no guidance for creating good videos – this remains future work. Instead, we compare ad-hoc videos with use cases as a widely used textual representation of requirements. Firstly, we compare the efficiency of creating videos and textual requirements descriptions by subjecting the analysts to time pressure. Secondly, we investigate the effectiveness, i.e. whether customers can distinguish valid from invalid requirements when they see them represented as use cases, or in videos. The ability to recognize requirements fast in a communication medium is a prerequisite for using that medium successfully during requirements analysis. The results of our experiment indicate that our subjects were able to capture more requirements with videos in the same time period. In contrast to others we find that videos can capture more requirements than use cases in the same time period.

This paper is organized as follows: In Sect. 2 we discuss possible situations where videos can be used in RE and describe the context of our investigations more closely. In Sect. 3 we give an overview of related work. In Sect. 4 we describe the design of our experiment based on the Goal-Question-Metric paradigm. Our results are presented in Sect. 5 and their validity is discussed in Sect. 6. Sect. 7 gives our conclusions and discusses questions for future research in video-based RE.

2 Video Opportunities in Requirements Engineering

In our experience, the benefit of using videos in RE depends on the state of requirements analysis and the type of stakeholder interaction (see Fig. 1):

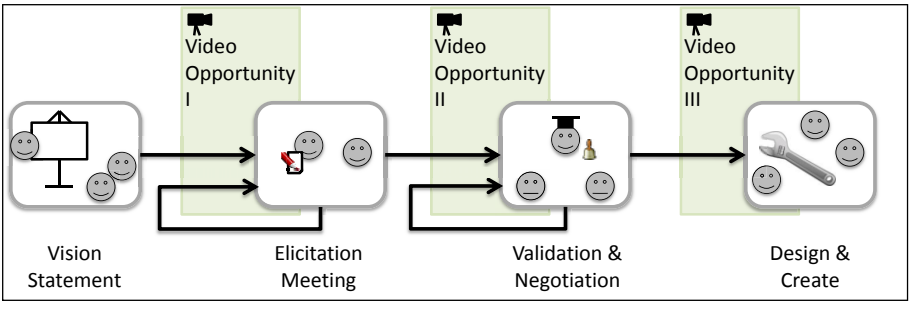


Fig. 1. Opportunities for using videos in Requirements Engineering

Videos in elicitation meeting: Analysts gain better understanding of the system under construction while planning potential use scenarios for videos and by enacting them. During the elicitation meeting, stakeholders can provide direct feedback on video scenes. However, very little is known about the system at this stage. The videos that can be created for elicitation meetings are based on rather abstract and visionary requirements, often derived from marketing [4] (i.e. from the vision statement). Hence, there is a high risk of creating irrelevant video scenes. Because of this risk, videos should not be too expensive, but focus on showing typical scenarios and contexts of usage for discussion with the stakeholders.

Videos for validation and negotiation: These videos are created based on the requirements from elicitation meetings. Requirements engineers have identified and interviewed stakeholders and interpreted their requirements in order to add concrete user goals to the project vision. During validation and negotiation, visualization of requirements in videos makes it easier for stakeholders to recognize their own requirements, and identify missing details or misinterpretations. Confirming recognized requirements and correcting or adding missing ones is equally important in this phase.

Videos in design and construction: Videos portrait the assumed application of a planned product. It contains assumptions on environmental conditions, the type of users envisioned, and their anticipated interaction with the system. This information is useful for system developers. There are approaches to enhance videos with UML models, which allow using videos as requirements models directly [2, 3]. In this work, we do not focus on videos in design and construction, but on videos in elicitation, validation, and negotiation meetings (video opportunities I and II in Fig. 1).

3 Related Work

Karlsson et al. illustrate the situation in market-driven projects [4]. Accordingly, in such projects developers invent the requirements. Since future customers of marked-driven products may not be available for elicitation, their feedback during validation will be especially important for project success. Simple techniques are needed that allow stakeholders to participate.

In this section we describe related work dealing with scenarios, requirements visualization, and videos as requirements representation techniques.

Scenario-based approaches. In order to support capturing, communicating and understanding requirements in projects, scenarios have been proposed by several researchers [1, 6, 7]. They allow analyzing and documenting functional requirements with a focus on the intended human-machine-interaction.

Anton and Potts show how to create scenarios systematically from goals [1]. They show that once a concrete scenario is captured other scenarios can easily be found.

There are several ways to document scenarios. One classic way is to create use cases that describe abstract scenarios [6]. The lack of concrete scenario representation is often observed as weakness, because it prevents stakeholders from understanding the specification. Therefore, Mannio and Nikula [7] describe the combination of prototyping with use cases and scenarios. They show the application of the method in a simple case study. In the first phase of their method, use cases are created and in one

of the later phases a prototype is constructed from multimedia objects like pictures, animations and sounds. The prototype is used to elicit the stakeholders' requirements. The prototyping leads to a more focused session.

A similar approach to support the creation of scenarios was presented by Maiden et al. [8, 9]: The ART-SCENE tool enables organizations to generate scenarios automatically from use cases and permits guided scenario walkthroughs. Zachos et al. propose to enrich such scenarios with multimedia content [10]. This facilitates recognizing and discovering new requirements. The evaluation of this approach is promising, but still additional evaluation is needed to understand the value of videos in RE.

Visualization of Requirements. Truong et al. describe storyboards as a technique to visualize requirements [22]. Storyboards are able to graphically depict narratives together with context. Williams et al. argue that a visual representation of requirements is important [11]. This visual representation makes RE more playful and enjoyable, thus contributing to better stakeholder satisfaction and more effective software development. They recommend using requirements in comic book style, because this allows combining visualizations with text. Williams et al. give no empirical evaluation if typical developers or stakeholders are able to create good-enough comic style requirements specifications. In addition, drawing comics may be time-consuming.

Videos in RE. Apart from comics, videos have been proposed as a good technique for adding visual representations to scenarios [2, 3, 12]. Broll et al. describe the use of videos during the RE (analysis, negotiation, validation, documentation) of two projects [12]. Their approach starts by deriving concrete contexts from general goals. Concrete scenarios are defined for each context of use. Based on these scenarios, videos are created. In parallel, the scenarios are analyzed. Both the video material and the analysis results are used to negotiate requirements in focus groups (small groups of important stakeholders). Broll et al. do not provide quantitative data about the effectiveness of their approach, but share qualitative lessons learned. They conclude that amateur videos created with household equipment are sufficient for RE-purposes. Based on their experience, they expect video production to be expensive due to the time-consuming recording of video and audio material. Therefore, they recommend to consider videos in RE as an option, but to keep the cost minimal. We agree that videos in sufficient quality can be created by a development team.

Brügge, Creighton et al. present a sophisticated high-end technique for video analysis of scenarios [2, 3]. Their approach starts with the creation of scenarios. Based on these scenarios, videos are created and refined to professional scenario movies. The Software Cinema tool allows enriching videos with UML models. This combination is a formal representation of requirements, useful for subsequent phases of the software development process (i.e. design). They found it feasible to negotiate requirements based on videos. However, they did not discuss whether videos are superior to textual representations of scenarios or not.

Compared to related work, this paper contributes by presenting empirical results from the comparison of videos and text based scenarios. In contrast to the expectations of others, our results suggest that videos can be produced faster and at lower effort than use cases during requirements analysis.

4 Experiment Design

Experiments in software engineering are difficult to design. There is a tension: Real situations are difficult and expensive to reproduce and compare. Very simple effects may be easier to observe and compare, but have little significance for practical applications. Zelkowitz et al. present several levels of empirical validation [13]. Anecdotal evidence is easy to capture, but insufficient to derive conclusive results. Rigid experiments might enable us to apply statistical methods, but controlling all threats to validity is hardly possible in any non-trivial set-up. In order to improve RE, a careful balance is needed.

Our experiment is designed to cover a few relevant aspects of working with videos, while simplify all others. Thus, it combines non-trivial real effects with the attempt to control threats to validity. The Goal-Question-Metric paradigm (GQM, see [14]) provides guidance for metrication in process improvement. It proposes a goal-oriented approach to selecting and defining metrics. Due to the real-world constraints (effort, limited comparability / repeatability), GQM is often applied to study phenomena in a rigid and disciplined way, without claiming statistical significance or generalizability.

We had eight student volunteers who had all some experience writing and reading use cases, but no or very limited experience using videos. None had applied videos to requirements before. Students had a computer science background and were in their second to fourth year. Two researchers acted as customers. Each of them adopted two tasks ("project requirements") and explained them to some of the students (see below). The first task was about navigating within the university building in order to find a person or office (person finder). The second task was about an airport check-in with the ability to assign waiting tickets to all boarding pass holders - and priority checking of late passengers who might otherwise miss their planes (adaptive check-in). Both customers were encouraged to invent more details and requirements about these visions. None of the subjects was involved in this research before. Customers did not know what the experiment was about before the evaluation started.

We use a short time slice for requirements elicitation. Use cases vs. ad-hoc videos are used to document elicited requirements, and to present them to the customers for validation. Counting recognized requirements is afforded by using lists of explicit requirements as a reference. In a pre-study, we examined feasibility of that concept [15]. Based on lessons learned in that pre-study, we made adjustments and refinements for the current experiment design. This new design makes best use of our available subjects in the above-mentioned context. We are aware of the threats due to student subjects and the limited number of repetitions, but consider those limitations acceptable [16] (see discussion in Sect. 6). We consider our experiment scenarios appropriate to represent the kind of real-world situations we want to study. They are relevant for evaluating the potential of videos in RE.

4.1 Goals of Investigation

GQM starts by looking at goals for improvement, and measurement goals. We stated goals of our investigation and used a number of cognitive tools and techniques to refine them into questions, hypotheses, and finally metrics that were designed into the experiment. At the same time, we took systematic precautions to limit and reduce

threats to validity. Other researchers are invited to use our considerations and design rationale as a starting point to replicate or extend our experiment. A replication in industry would be expensive, but particularly welcome. Our questionnaire and experiment kit are available for replications of our study.

Goal of investigation:
Investigate effectiveness and efficiency of creating ad-hoc videos under time pressure for validation of early requirements compared to use cases

Goal 1: Analyze effectiveness and efficiency of use cases (…)
Goal 2: Compare effectiveness and efficiency of videos with respect to use cases (…)
Goal 3: Analyze subjective preferences of videos with respect to use cases (…)

For each goal, a number of characterizing facets were specified. According to GQM [14] and our own experience in applying it in industry [17], this explicit facet classification helps to focus measurement and to avoid ambiguities.

Table 1. Facet classification of the three goals of our investigation

	Purpose	**concerning aspect**	**of object**	**in context**	**from perspective**
Goal 1	Analyze	Effectiveness and efficiency	Use cases	representing requirements for validation under time pressure	Customer
Goal 2	Compare	Effectiveness and efficiency	Ad-hoc videos with respect to use cases		Customer
Goal 3	Analyze	Preferences	Ad-hoc videos with respect to use cases		Requirements engineers

In the pre-study, we had analyzed both customer and developer perspectives and what they recognized. They represent requirements analysis and design&create tasks. In this paper, only the customer is defined to be the reference for recognizing requirements. Requirements that are not perceived by the customer cannot be confirmed or corrected during validation. Therefore, the customer perspective is adopted for comparing effectiveness and efficiency for goals 1 and 2 in Table 1. When personal preferences are solicited for goal 3, however, we focus on the requirements engineers' perspective: videos only deserve further investigation if requirements engineers accept them and consider them useful. Similar considerations are stimulated by the other facets. For example, the purpose of comparing things (goal 2) requires a reference for that comparison – we planned to apply use cases analyzed in goal 1 for that purpose.

4.2 Research Questions and Hypotheses

In order to reach the goals, a number of questions need to be answered: This is how our research questions are related to the above-mentioned goals of our investigation. According to GQM, goals are further refined into questions and hypotheses. Abstraction sheets [18] may be used to guide the refinement and preparation process. They force researchers to make decisions on details of their research questions.

In goals 1 and 2, "effectiveness and efficiency" are mentioned. We defined effectiveness and efficiency as "*representing many requirements*" and "*representing requirements fast*", respectively. Since the context of all three above-mentioned goals is specified as "*validation of requirements under time pressure*", we decided to merge both efficiency and effectiveness in this context into "*representing many requirements in a limited amount of time*".

During validation, detecting a misunderstanding or an error is as valuable as confirming a correctly recognized requirement. Therefore, we are interested in the total number of *issues* discussed when stimulated by videos (or use cases). It is important that we know more about an issue afterwards.

In order to make this quality aspect measurable in the experiment, we define two research questions based on Kano's widely-known classification of requirements [19]:

- How many of his or her basic/performance/excitement requirements can a customer recognize in a given set of use case vs. ad-hoc videos during validation?
- How many errors would a customer detect if use cases vs. videos were created and presented under time pressure?

According to Kano, we distinguish between basic, performance, and excitement requirements. Basic requirements tend to be neglected and overlooked as "trivial". Performance requirements are normal requirements that are most likely to be discussed explicitly. Excitement requirements are unconscious requirements. When they are fulfilled, customers can get excited. However, implementing a misinterpreted requirement might have the opposite effect.

The rationale for referring to Kano categories (basic/performance/excitement) is based on **hypotheses** that were raised by pre-study results:

a. Customers will recognize a similar amount of performance requirements in both techniques [estimate: +/-10% (#req(use case) - #req(use case)/10 < #req(video) < #req(use case) + #req(use case)/10].
b. Customers will be able to identify more errors and problems concerning basic requirements in videos than in use cases [estimate: >50% more]
c. For early requirements on an innovative product, customers will be stimulated to identify more excitement requirements (correct or false) in videos than in use cases – when both are built under comparable time pressure [estimate: 1 or 2 excitement requirements with use cases, at least 3 with videos].

While (a) was directly observed in the pre-study, (b) is based on the concrete nature of videos: Even seemingly "trivial" (basic) requirements must be represented somehow in a video, while use cases may simply abstract from details or ignore them. Hypothesis (c) builds on the assumption that a playful multimedia representation such as videos invites and encourages more exciting interpretations on both the requirements engineers' and the customer's parts. Interestingly, both confirmed requirements and uncovered deviations are considered a success during validation: Both contribute to better mutual understanding and fewer remaining misunderstandings. Estimations in [parentheses] give a quantitative idea of the effect we expected. GQM requires such estimates in order to interpret finally measured results. Without estimates, many practitioners and researchers tend to think they "knew this before" - in hindsight. A concrete estimate serves as a reference; of course, one must not change estimates after

seeing actual results. Since most measurements in real-world settings do not provide statistically significant results, it is even more important for interpretation of results to define what we mean by "similar", "more" and "remarkably more", respectively.

Goal 3, asking for the subjective preferences of our subjects, was evaluated using a questionnaire. Basically we asked whether our subjects preferred videos or use cases for documentation under time pressure.

4.3 Preparing Metrics for the Experiment

Based on explicit questions and hypotheses, metrics can be selected or defined. GQM is often applied to measuring symptoms of process improvement in real-world environments [17]. In those cases, metrics should be integrated into existing practices; this reduces measurement effort and mitigates the risk to distort the measured object by the measurement.

Our experiment is designed to reflect non-trivial real-world aspects of validation under time pressure, and to accommodate measurement at the same time. In order to fully exploit the available resources (participants, time, effort), experiment tasks were carefully designed. Measurement is supported by forms and a questionnaire for goal 3: subjective preference. When GQM is applied consistently, metrics results directly feed back into evaluation and interpretation. The overall setup is shown in Table 2.

Table 2. Setup of experiment using eight subjects (a-h) and two customers (A, W)

		Tasks / projects				
			person finder	***adaptive check-in***		
Customer A	config. 1	a,b	use cases	video	e,f	config. 2
		c,d	videos	use cases	g,h	
Customer W	config. 3	e, f	use cases	videos	a,b	config. 4
		g, h	videos	use cases	c,d	

Chronological sequence: phase 1 phase 2

Two researchers (A, W) acted as customers. Each phase contained two configurations, one for each customer. A configuration is characterized by the same task, the same customer, and two pairs of subjects working independently, one applying use cases, the other applying videos. In the second phase, a different task was presented by a different customer and the pairs of subjects exchanged techniques. Each configuration followed the same schedule:

10 min. Customer explains task to all subjects of that configuration together (e.g., a, b, c, d). They may ask clarification questions.

30 min Pairs of subjects conceive and produce use cases vs. videos. *In parallel*, customers write down a list of those requirements that were explicitly mentioned during the 10 minute slot of explanation and questions.

10 min.	Pairs clean up their results. They rewrite text and use cases, download video clips from the cameras, rename files etc.
Afterwards	Each customer evaluates use cases and videos with respect to the reference list of explicit requirements (see above). They check and count all recognized requirements, and count false or additional requirements raised. They use a form to report those counts.

Fig. 2 shows three excerpts from different videos produced during the experiment. All teams used hand-drawn paper mockups, combined them with available hardware like existing info terminals or furniture in the university building, or mobile phones. By interacting with new, pretended functionality, subjects illustrated the envisioned system. Most groups enacted scenarios like a passenger in a hurry who benefits from an advanced ticket system (Fig. 2, center).

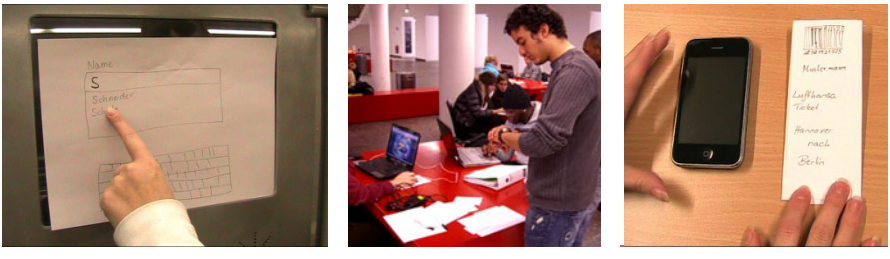


Fig. 2. Mockup screen, check-in of passenger in a hurry, advanced eTicket

4.4 Rationale of Experiment Design

In the pre-study, we wanted to explore the feasibility of using video for fast requirement documentation. With respect to Table 2, the pre-study resembled one configuration, but with four people interpreting the results of that one configuration.

We wanted to repeat the experiment in order to substantiate our observations. We were able to add four configurations. Given our research questions, we needed to investigate the validation capabilities of videos vs. use cases in more detail. It was not sufficient to recognize intended requirements in the use cases or videos – we wanted to classify represented requirements based on Kano's categories [19].

Design inspired by factorial variation can be used in software engineering in order to exploit the scarce resource of appropriate subjects better, and to avoid threats associated with dedicated control groups: All eight subjects carried out two tasks in two subsequent phases. We varied tasks, customers, and media in a systematic way in order to improve control of potential influence factors. This design reduces undesired learning effects and biases for a particular technique. Variation of techniques, customers, and tasks was used to minimize threats to validity (see Sect. 6).

Pairing subjects has a number of advantages: (a) The ability to communicate in a pair improves the chance to derive ideas and reflect on them. (b) Two people can do more work than individuals: write more use cases and make videos of each other, which amplifies the visible impact of both techniques. (c) Different personalities and their influence should be averaged out in pairs as opposed to individuals. We

considered those aspects more relevant than a higher number of configurations containing only individuals instead of pairs.

5 Results

The counts of requirements recognized, and of additional requirements identified on basic/excitement level are indicated in Table 3. The customer in a configuration provided the reference for "requirements explicitly stated" during the first 10 minute slot of explanations and questions. All recognized and represented requirements were marked in that list by their respective customer based on an audio recording of the explanation session.

Table 3 presents all use case pairs in the top part, and their corresponding video pairs in the lower part. The right-hand columns provide the average of additionally raised requirements and the average percentage of recognized requirements. As additional requirements we count new basic or excitement requirements were raised and were confirmed or corrected by the customer. Both types (corrected, confirmed) are requirements that otherwise would have been forgotten. The sum of requirements confirmed and corrected is given below the category (basic, excitement).

Table 3. Results of experiment: counts and percentages (average over all configs.)

			Config 1	Config 2	Config 3	Config 4	Avg. absolute	Avg. % of total
customer	**total reference**	on explicit req. list	**15**	**17**	**31**	**16**		
use case	**recognized**	**performance reqs.**	**10**	**7**	**20**	**9**		**57%**
		basic reqs. confirmed	1	1	1	0	*0,8*	
		basic reqs. corrected	0	1	3	1	*1,3*	
	confirmed+corrected	**basic reqs.**	**1**	**2**	**4**	**1**	***2,0***	
		excitem. confirmed	0	2	2	1	*1,3*	
		potential exc. corrected	1	1	2	1	*1,3*	
	confirmed+corrected	**excitement reqs.**	**1**	**3**	**4**	**2**	***2,5***	
video	**recognized**	**performance reqs.**	**14**	**12**	**20**	**11**		**74%**
		basic reqs. confirmed	2	5	0	1	*2,0*	
		basic reqs. corrected	0	1	1	1	*0,8*	
corresponds	confirmed+corrected	**basic reqs.**	**2**	**6**	**1**	**2**	***2,8***	
to use case		excitem. confirmed	0	3	1	0	*1,0*	
in same		potential exc. corrected	0	0	3	0	*0,8*	
column	confirmed+corrected	**excitement reqs.**	**0**	**3**	**4**	**0**	***1,8***	

Configuration: same task, same customer, same phase

The answers to questions regarding goal 3 (subjective preferences of requirements engineers) were solicited using a questionnaire. Part of that questionnaire was completed before the experiment started (competencies, previous experience with video, year of study etc.). The evaluative questions were asked after all four configurations had produced their results, but before they were published or analyzed. Due to the design of our experiment, each subject had carried out one video and one use case task, in different orders. Customers were not told the details, hypotheses, or implications of the experiment. Questionnaires were completed by all eight subjects. This small number may limit the statistical power. For that reason, only absolute numbers are given below. According to GQM, asking for subjective judgement is a legitimate type of metrics, if one wants to draw conclusions on subjective opinions. It would be

much more difficult, artificial, and error-prone to distil satisfaction and preference data from "objective observations".

We asked for potential preferences of use cases over videos:

- Subjects considered videos more appropriate for an overview. They appeared less ambiguous and better suited to illustrate functions. Use cases were preferred to document exceptional behaviour and alternative steps. Pre- and postconditions were mentioned as advantages, together with a finer level of detail.
- Under time pressure, 7 (total: 8) subjects would prefer videos for documenting requirements for various reasons: better description of requirements (6), better coverage of usability aspects (6), more functional requirements per time (3), or generally more requirements per time (2).
- Without time pressure, still 5 (total: 8) subjects would prefer videos for documenting requirements; only 2 would prefer use cases.

5.1 Interpretation of Results

When GQM is used with explicit expectations, results can be easily compared to the above-mentioned hypotheses. The most promising expectations and respective actual results are briefly commented:

- "Customers will recognize a similar amount of performance requirements in both techniques [estimate: +/-10% (#req(use case) - #req(use case)/10 < #req(video) < #req(use case) + #req(use case)/10]."
    - Customers recognized 57 % of their requirements in use cases and 74% in videos.
    - Although this difference is far higher than the 10% expected, our small number of configurations (4) limits statistical power and significance.
    - Since there was no configuration in which use cases performed better than videos, the four configurations support the feasibility of video-based requirements validation.
- "Customers will be able to identify more errors and problems concerning basic requirements in videos than in use cases [estimate: >50% more]"
    - Use cases led to an average of 2.0 additional basic requirements being confirmed or corrected. In comparison, videos raised 2.8.
    - Therefore, videos led to 40 % more additional basic requirements than use cases. Our expectation of more than 50% is not fulfilled.
    - Nevertheless, the experiment has approved the tendency that videos raise more basic requirements than use cases.
- "For early requirements on an innovative product, customers will be stimulated to identify more excitement requirements (correct or false) in videos than in use cases – when both are built under comparable time pressure [estimate: 1 or 2 excitement requirements with use cases, at least 3 with videos]."
    - Use cases stimulated an average of 2.5 excitement requirements, videos performed slightly worse at an average of 1.8.
    - Our expectation is not supported by the observations and counts. Use cases scored higher in two of the configurations, in the other two configurations videos and use cases raised the same amount.

- We conclude that we cannot support this hypothesis. Our assumption may be wrong: Using an "innovative technique" (videos) for RE does not necessarily imply a higher rate of creative ideas.

These results respond directly to our questions, which are repeated above. We wanted to analyze selected aspects of use cases, and compare them to videos. Our interpretation of results responds to the goals and associated questions in a detailed and well-defined way. All together, we conclude the analysis and comparison:

- Videos could be used in all four configurations. They did not fail or perform drastically worse in any configuration.
- Their overall performance of making requirements and errors recognizable was better for explicit requirements.
- For basic requirements our expectations are supported in tendency. In case of the excitement requirements our expectations were not met. Instead, use cases led to more additional excitement requirements than videos.
- Our subjects preferred videos over use cases and assumed they could help to find and validate more requirements. They did not yet know experiment results.

Although we did not expect to find statistically significant differences, our experiment shed more light on a situation and a technology (video) that is frequently mentioned or applied without any empirical evidence.

6 Discussion of Validity

Wohlin emphasizes the necessity to consider threats to validity during experiment planning [20]. In Sect. 4, the design of our experiment referred to several threats to validity - and provides rationale how we tried to avoid them. Nevertheless, several threats remain and should be considered when our results are used.

Our "treatment" is the application of either use cases or videos to the representation of requirements. We discuss four types of validity in the order of descending priority that Wohlin considers appropriate for applied research:

Internal validity *(do we really measure a causal relationship between videos and requirements in validation?)*: We paired subjects randomly under the constraint that each pair included one student of computer science and one of mathematics. We took care to build equally strong pairs. The cross-design shown in Table 2 was inspired by Basili et al. [21] in order to compensate for previous knowledge. Then each pair used the other technique to compensate for learning during the experiment. Volunteers are said to be a threat to validity since they tend to be more motivated and curious than regular employees. This may be true, but our target population may be close enough to our subjects to reduce that threat (see external validity).

There is a threat to validity we consider more severe: When customers evaluate results (use cases and videos) for "recognized requirements" and additional findings, their judgment should be as objective and repeatable as possible. We took great care to handle this threat: A customer audio-recorded the 10 minute explanation session and derived the list of 15-31 requirements that were explicitly raised during the explanation or questions. When customers evaluated results, they used this list as a checklist, which makes the evaluation process more repeatable. Obviously, the granularity of what was considered "one" requirement is very difficult and might

cause fierce discussions among any two requirements experts. Our attempt to cope with this threat is using "configurations" in which two pairs (one use case, one video) operate under the same conditions, no matter what those conditions might be in detail: same customer, same granularity, same task, participated in same 10-minute session with same questions asked. By using four configurations, we try to compensate for random influences in a given situation.

External validity *(are the findings valid for the intended target population?):* Students and volunteers are usually regarded poor representatives of industry employees [16]. However, our work tries to support the upcoming generation of requirements engineers who are familiar with video-equipped mobile phones and multimedia handhelds. As explained in [15], two new developments encouraged us to explore ad-hoc video for requirements validation in the first place: (1) the advent of inexpensive, ubiquitous recording devices and (2) a generation of requirements engineers that have grown up using mobile phones and PDAs voluntarily in their private life. Today's (high school and) university students might represent that generation even better than current industry employees who learned to practice RE with DFDs, Doors etc. All of our subjects had completed at least one lecture that included a substantial portion (8h lecture time) of "traditional" RE. We consider our students valid representatives of upcoming requirements engineers in practice - which they may actually become in a year or two.

Construct validity *(did we model the real phenomena in an appropriate way?):* A major threat to construct validity is a poor understanding of the questions and concepts examined. This can lead to misguided experiment design. By following GQM carefully, we were forced to specify our goals, questions, and derived metrics in detail. For example, specifying purpose and perspective as goal facets usually helps clarifying aspects that are otherwise neglected as "trivial". The GQM technique guided us from the goal of investigation down to the form used by customers to mark "recognized explicit requirements", and additional findings in the "explicit reqs. list".

Conclusion validity *(What is the statistical power of our findings?):* Conclusion validity is considered lowest priority for applied research by Wohlin et al. [20] - large numbers of subjects and statistical significance are very difficult to get in a real or realistic setup. GQM is a technique optimized for exploring effects in practice, not so much for proving them statistically [17]. Nevertheless, even in our small sample of eight projects (4 tasks* 2 pairs), some differences are large enough to reach statistical significance: e.g., the recognized number of explicit requirements is higher with videos than with use cases (statistically significant at alpha=10% in a double-sided paired t-test). Although the statistical power is not very high, (p=0.86), an effect that even reaches statistical significance is the exception rather than the rule in GQM.

7 Conclusion and Outlook

Software systems have become a ubiquitous part of everyday life. Many aspects of modern society rely on mobile phones, PDAs, and sophisticated devices that interact with each other and support processes via software. Check-in procedures at airports, personal navigation solutions, and numerous other applications are envisioned and developed for a growing market. In many cases, those visions need to be turned into products rather fast, in order to keep a competitive edge.

However, traditional RE techniques have not yet embraced the opportunities of ubiquitous modern technology, such as mobile devices and video cameras. The generation of future requirements engineers (i.e., current university students) grows up with the technical ability to record ad-hoc videos at almost no cost or effort. We try to benefit from these new opportunities, and enhance requirements documentation and validation under time pressure by using ad-hoc videos.

Our pre-study encouraged us to consider videos a feasible option, compared to use cases. However, this single experiment needed to be replicated - and enriched to explore how videos affect the recognition of different Kano types of requirements. Our experiment was designed to reduce threats to validity. We followed GQM in order to get plausible and reliable results despite the small number of subjects (8), and the remaining threats to validity - which is difficult to avoid even in a small validation setup. Results showed a higher recognition rate for performance requirements, and a higher rate of identifying basic requirements. To our surprise, excitement requirements were not confirmed or falsified at a higher rate than with use cases.

We explored whether videos could make a contribution to coming RE techniques. For the experiment reported above, we wanted to single out and highlight differences (including time pressure). For example, one will not use videos alone in an industrial environment. We develop specific tools for handling video clips, and combining them with manual sketches, still pictures - and use cases. Despite the many threats to validity, our pre-study and the results presented above indicate that there is good reason to take videos seriously. In contrast to expectations in related work, videos can capture more requirements per time period than use cases. Still, further studies and concepts are needed to fully integrate them in future RE.

In this experiment, pure use of videos was investigated; in a real project one would combine the advantages of traditional and innovative techniques. For example, we develop tools to support handling of video clips, and comparing variants easily. Integrating sketches and pictures into a video, as well as controlling scenarios by use case steps are more complex examples of supporting requirements validation.

In our future work, we will continue to explore the impact of new opportunities in RE by experiments. Those opportunities may be exploited by developing adapted procedures and supportive tools. When the world of ubiquitous applications changes fast, feedback and validation must exceed traditional channels. Videos seem to be a viable option, as our experiment shows.

References

1. A.I., Potts, C.: The use of goals to surface requirements for evolving systems. In: ICSE 1998: Proceedings of the 20th International Conference on Software Engineering, Leipzig, Germany, pp. 157–166. IEEE Computer Society, Los Alamitos (1998)
2. Creighton, O., Ott, M., Bruegge, B.: Software Cinema-Video-based Requirements Engineering. In: RE 2006: Proceedings of the 14th IEEE International Requirements Engineering Conference, Minneapolis, Minnesota, USA, pp. 106–115. IEEE Computer Society, Los Alamitos (2006)
3. Bruegge, B., Creighton, O., Reiß, M., Stangl, H.: Applying a Video-based Requirements Engineering Technique to an Airport Scenario. In: MERE 2008: Proceedings of the 2008 Third International Workshop on Multimedia and Enjoyable Requirements Engineering, Barcelona, Katalunya, Spain, pp. 9–11. IEEE Computer Society, Los Alamitos (2008)

4. Karlsson, L., Dahlstedt, Å.G., Nattoch Dag, J., Regnell, B., Persson, A.: Challenges in Market-Driven Requirements Engineerng - an Industrial Interview Study. In: Proceedings of Eighth International Workshop on Requirements Engineering: Foundation for Software Quality, Essen, Germany (2002)
5. Fischer, G.: Symmetry of Ignorance, Social Creativity, and Meta-Design. In: Creativity and Cognition 3 - Intersections and Collaborations: Art, Music, Technology and Science, pp. 116–123 (1999)
6. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley Professional, Reading (January 2000)
7. Mannio, M., Nikula, U.: Requirements Elicitation Using a Combination of Prototypes and Scenarios. In: WER, pp. 283–296 (2001)
8. Zachos, K., Maiden, N., Tosar, A.: Rich-Media Scenarios for Discovering Requirements. IEEE Software 22, 89–97 (2005)
9. Seyff, N., Maiden, N., Karlsen, K., Lockerbie, J., Grünbacher, P., Graf, F., Ncube, C.: Exploring how to use scenarios to discover requirements. Requir. Eng. 14(2), 91–111 (2009)
10. Zachos, K., Maiden, N.: ART-SCENE: Enhancing Scenario Walkthroughs With Multi-Media Scenarios. In: Proceedings of Requirements Engineering Conference (2004)
11. Williams, A.M., Alspaugh, T.A.: Articulating Software Requirements Comic Book Style. In: MERE 2008: Proceedings of the 2008 Third International Workshop on Multimedia and Enjoyable Requirements Engineering, Barcelona, Catalunya, Spain, pp. 4–8. IEEE Computer Society, Los Alamitos (2008)
12. Broll, G., Hußmann, H., Rukzio, E., Wimmer, R.: Using Video Clips to Support Requirements Elicitation in Focus Groups - An Experience Report. In: 2nd International Workshop on Multimedia Requirements Engineering (MeRE 2007), Conference on Software Engineering (SE 2007), Hamburg, Germany (2007)
13. Zelkowitz, M.V., Wallace, D.R.: Experimental validation in software engineering. Information & Software Technology 39(11), 735–743 (1997)
14. Basili, V.R., Caldiera, G., Rombach, H.D.: The Goal Question Metric Approach. In: Encyclopedia of Software Engineering, pp. 646–661. Wiley, Chichester (1994)
15. Schneider, K.: Anforderungsklärung mit Videoclips. In: Proceedings of Software Engineering 2010, Paderborn, Germany (2010)
16. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in Using Students in Empirical Studies in Software Engineering Education. In: METRICS 2003: Proceedings of the 9th International Symposium on Software Metrics, Sydney, Australia. IEEE Computer Society, Los Alamitos (2003)
17. Schneider, K., Gantner, T.: Zwei Anwendungen von GQM: Ähnlich, aber doch nicht gleich. Metrikon (2003)
18. van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development. McGraw-Hill Publishing Company, New York (1999)
19. Kano, N.: Attractive Quality and Must-be Quality. Journal of the Japanese Society for Quality Control, 39–48 (1984)
20. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C.: Experimentation In Software Engineering: An Introduction, 1st edn. Springer, Heidelberg (1999)
21. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The Empirical Investigation of Perspective-Based Reading. Int. Journal of Empirical Software Engineering 1(2), 133–164 (1996)
22. Truong, K.N., Hayes, G.R., Abowd, G.D.: Storyboarding: An Empirical Determination of Best Practices and Effective Guidelines. In: DIS 2006: Proceedings of the 6th Conference on Designing Interactive Systems, Pennsylvania, USA, pp. 12–21. ACM, New York (2006)

Supporting the Consistent Specification of Scenarios across Multiple Abstraction Levels

Ernst Sikora, Marian Daun, and Klaus Pohl

Software Systems Engineering
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen, 45117 Essen
{ernst.sikora,marian.daun,klaus.pohl}@sse.uni-due.de

Abstract. [**Context and motivation**] In scenario-based requirements engineering for complex software-intensive systems, scenarios must be specified and kept consistent across several levels of abstraction such as system and component level. [**Question/problem**] Existing scenario-based approaches do not provide a systematic support for the transitions between different abstraction levels such as defining component scenarios based on the system scenarios and the system architecture or checking whether the component scenarios are consistent with the system scenarios. [**Principal ideas/results**] This paper presents a systematic approach for developing scenarios at multiple abstraction levels supported by automated consistency checks of scenarios across these abstraction levels. [**Contribution**] We have implemented the consistency check in a tool prototype and evaluated our approach by applying it to a (simplified) adaptive cruise control (ACC) system.

Keywords: abstraction levels, consistency, interface automata, scenarios.

1 Introduction

Scenario-based requirements engineering (RE) is a well proven approach for the elicitation, documentation, and validation of requirements. In the development of complex software-intensive systems in the embedded systems domain, scenarios have to be defined at different levels of abstraction (see e.g. [1]). We call scenarios that specify the required interactions of a system with its external actors "system level scenarios" and scenarios that additionally define required interactions between the system components "component level scenarios". For brevity, we use the terms "system scenarios" and "component scenarios" in this paper.

Requirements for embedded systems in safety-oriented domains such as avionics, automotive, or the medical domain, must satisfy strict quality criteria. Therefore, when developing system and component scenarios for such systems, a rigorous development approach is needed. Such an approach must support the specification of scenarios at the system and component level and ensure the consistency between system and component scenarios. Existing scenario-based approaches, however, do not provide this kind of support.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 45–59, 2010.

In this paper, we outline our approach for developing scenarios for software-intensive systems at multiple abstraction levels. This approach includes a methodical support for defining scenarios at the system level, defining component scenarios based on the system scenarios and an initial system architecture as well as detecting inconsistencies between system scenarios and component scenarios. Our consistency check reveals, for instance, whether the component scenarios are complete and necessary with respect to the system scenarios. To automate the consistency check, we specify system and component scenarios using a subset of the message sequence charts (MSC) language [2]. The consistency check is based on a transformation of the specified MSCs to interface automata [3] and the computation of differences between the automata or, respectively, the regular languages associated with the automata. We have implemented the consistency check in a prototypical tool and evaluated our approach by applying it to a (simplified) adaptive cruise control (ACC) system.

The paper is structured as follows: In the remainder of this section, we provide a detailed motivation for our approach. Section 2 outlines the foundations of specifying use cases and scenarios using message sequence charts. Section 3 briefly describes our approach for specifying system and component scenarios. Section 4 provides an overview of our technique for detecting inconsistencies between system and component scenarios. Section 5 summarises the results of the evaluation of our approach. Section 6 presents related work. Section 7 concludes the paper and provides a brief outlook on future work.

1.1 Need for Specifying Requirements at Different Abstraction Levels

Abstraction levels are used to separate different concerns in systems engineering such as the concerns of a system engineer and the concerns of component developers. We illustrate the specification of requirements at multiple levels of abstraction by means of an interior light system of a vehicle. At the system level, the requirements for the interior light system are defined from an external viewpoint. At this level, the system is regarded as a black box, i.e. only the externally visible system properties are considered without defining or assuming a specific internal structure of the system. For instance, the following requirement is defined at the system level:

– *R2 (Interior light system)*: The driver shall be able to switch on the ambient light.

The level of detail of requirement R2 is typically sufficient for communicating about the requirements with system users. However, for developing and testing the system, detailed technical requirements are needed. To define these detailed technical requirements, the system is decomposed into a set of architectural components and the requirements for the individual components and the interactions between the components are defined based on the system requirements. For instance, the following requirements are defined based on requirement *R2* (after an initial, coarse-grained system architecture has been defined for the interior light system):

– *R2.1 (Door control unit)*: If the driver operates the ‘Ambient’ button, the door control unit shall send the message LIGHT_AMB_ON to the roof control unit.
– *R2.2 (Roof control unit)*: If the roof control unit receives the message LIGHT_AMB_ON, it shall set the output DIG_IO_AMB to ‘high’.

1.2 Need for Checking Requirements Consistency across Abstraction Levels

If the component requirements define, for instance, an incomplete refinement of the system requirements, the integrated system will not meet its requirements even if each component satisfies the requirements assigned to it. For example, if the requirement *R2.2* in the previous section was omitted, the door control unit would send the activation signal to the roof control unit, yet the roof control unit would not be able to process this signal and hence it would not switch on the light.

If the system components are developed by separate development teams or even by separate organisations, specification errors such as inconsistencies between system and component requirements may remain hidden until very late stages of the development process, typically until system integration. To avoid rework during system integration caused by such defects (which often leads to project delays), requirements engineers must check early in the development process whether the component requirements are consistent with the system requirements. In addition, for safety-relevant systems, a proof must be provided that each component requirement is necessary (see e.g. [4]). The necessity of a component requirement can be shown by demonstrating that this requirement is needed to satisfy a system requirement.

1.3 Main Objectives of the Scenario-Based RE Approach

Based on the above considerations, the following objectives for a scenario-based approach can be defined:

- *O1: Specification of system scenarios*. The approach should support the specification of scenarios at the system level. For this purpose, it should provide guidelines defining what kind of information should be contained in the system scenarios. The way the system scenarios are specified should ease the transition to component scenarios as well as consistency checking.
- *O2: Specification of component scenarios*. The approach should support the specification of component scenarios based on the system scenarios and a coarse-grained architecture. For this purpose, it should provide guidelines defining what kind of information is added in the component scenarios. Furthermore, the approach should allow structuring the component scenarios differently from the system scenarios, for instance, to improve readability of the scenarios.
- *O3: Consistency checking of system and component scenarios*. The approach should support checking whether the externally visible system behaviour defined at the system level and the one defined at the component level conforms to a defined (possibly project-specific) consistency criterion. To support the removal of detected inconsistencies, the approach should provide a detailed account of all detected differences between the system and component scenarios.

2 Specification of Scenarios Using Message Sequence Charts

A scenario documents a sequence of interactions leading to the satisfaction of a goal of some agent (see e.g. [5]). Multiple scenarios associated with the same goal or set of

goals are typically grouped into use cases (see e.g. [6]). Scenarios can be documented using various formats such as natural language, structured templates, or diagrams.

To facilitate automated verification, we use message sequence charts (see [2]) for specifying and grouping scenarios, both, at the system and the component level. We decided to use message sequence charts since they are commonly known in practice and offer a standardised exchange format. The specification of scenarios using message sequence charts is outlined in this section. The formalisation of message sequence charts employed in our approach is outlined in Section 4.

2.1 Basic and High-Level Message Sequence Charts

The message sequence charts language defines basic message sequence charts (BMSCs) and high-level message sequence charts (HMSCs). The essential elements of a BMSC are instances and messages (see Fig. 1). The essential elements of a HMSC are nodes and flow lines. A node can refer to a BMSC or another HMSC. A flow line defines the sequential concatenation of two nodes. Therein, the sequential order of the nodes may contain iterations and alternatives. Formal definitions of the (abstract) syntax of BMSCs and HMSCs are given, for instance, in [7]. The graphical notation of BMSCs and HMSCs used in this paper is shown in Fig. 1.

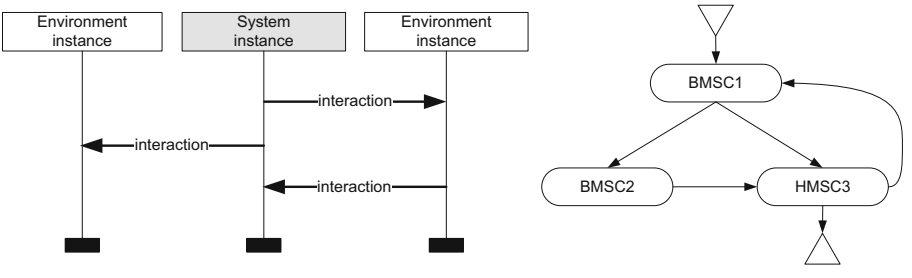


Fig. 1. Graphical notation of BMSCs (left-hand side) and HMSCs (right-hand side)

2.2 Specifying Use Cases and Scenarios Using Message Sequence Charts

Fig. 2 shows the documentation and composition of scenarios using message sequence charts as opposed to the documentation and grouping of scenarios by means of use case templates (see e.g. [6]). In our approach, we use message sequence charts in order to reduce the ambiguity caused by documenting scenarios and their composition using natural language. We use BMSCs for documenting atomic scenarios (or scenario fragments) and HMSCs for scenario composition. Therein, a HMSC can compose several scenario fragments into a larger scenario, several scenarios into a use case, or several use cases into a larger use case (Fig. 2, right-hand side). By using HMSCs, relationships between use cases such as “include” and “extend” as well as the sequential execution of use cases can be defined. A similar approach based on UML activity diagrams and UML sequence diagrams is described in [8]. Note that other information such as use case goals or pre- and post-conditions still need to be documented in the use case template.

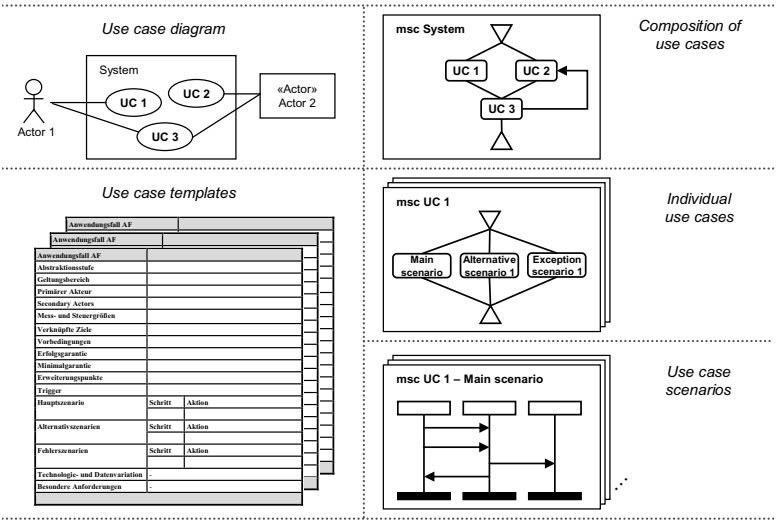


Fig. 2. Templates (left) vs. messages sequence charts (right) for documenting scenarios

3 Specification of Scenarios at Two Abstraction Levels

Our overall approach consists of three main activities (see Fig. 3). We outline these three activities and their major inputs and outputs in the following subsections.

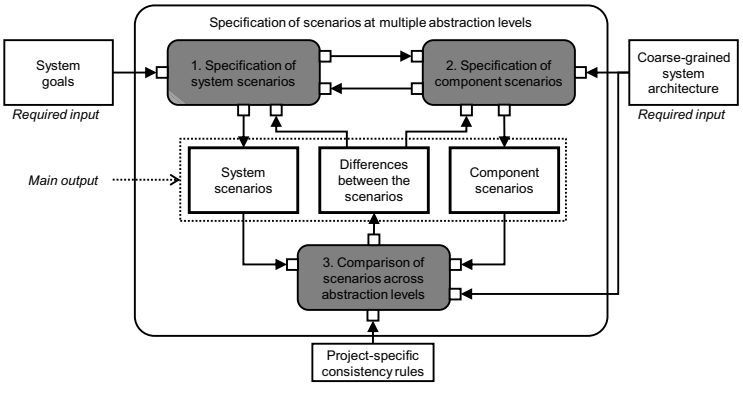


Fig. 3. Overall approach for specifying scenarios at multiple levels of abstraction

3.1 Specification of System Scenarios and Use Cases

The specification of system scenarios includes specifying individual system scenarios using BMSCs and interrelating the individual scenarios using HMSCs. Systems scenarios are identified and defined based on the goals of the external system actors. A system scenario should document the major interactions between the system and its

environment required to satisfy the associated goal. For each system goal, the relevant system scenarios satisfying this goal should be documented.

When specifying the system scenarios, the requirements engineers need to ensure that a black box view of the system is maintained. In other words, the specified system scenarios should only define the external interactions of the system and no internal interactions since defining the internal interactions is the concern of lower abstraction levels. Furthermore, system scenarios should be defined at a logical level, i.e. independently of a specific implementation technology such as a specific interface design or a specific system architecture. Fig. 4 shows a simple system scenario for the interior light system example introduced in Section 1.1.

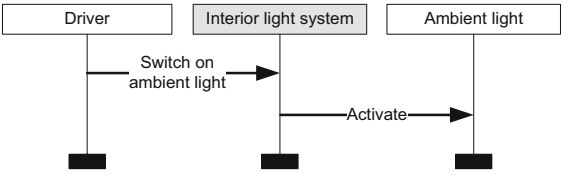


Fig. 4. Simple example of a system scenario

In early phases of use case development, the focus should be placed on the main or normal course of interactions that is performed to satisfy the use case goal(s). In later stages, alternative and exception scenarios should be added and related to the main scenario. However, in order to maintain the black box view, alternative and exception scenarios should be defined only if they are required independently of the internal structure of the system and the technology used for realising the system.

3.2 Specification of Component Scenarios and Use Cases

The definition of component scenarios comprises the definition of BMSCs and HMSCs at the component level. Typically, one starts defining component scenarios based on the defined system scenarios taking the (coarse-grained) system architecture into account. Furthermore, additional scenarios can be defined at the component level which are not based on the system scenarios such as scenarios for error diagnosis.

3.2.1 Definition of Component Scenarios Based on System Scenarios

A component scenario can define a possible realisation of a system scenario. For this purpose, the component scenario must define the interactions among the system components required to realise the interactions with the environment defined in the system scenario. The refinement of an individual system scenario is accomplished by the following steps (see e.g. [9]):

- *Step 1*: Identification of the components that are responsible for realising the system scenario. The instance representing the system in the system scenario is replaced by the identified set of components in the component scenario.
- *Step 2*: Each scenario step (i.e. each system-actor interaction) defined in the system scenario is assigned to a component. Therein, either the names of the scenario steps defined in the system scenario remain unchanged, or a unique mapping between

the message names is established. This is a necessary condition to facilitate consistency checking across abstraction layers.
– *Step 3*: The component scenario is completed by adding the required, system-internal interactions (or signal flows) between the system components.

A simple example of a component scenario is depicted in Fig. 5.

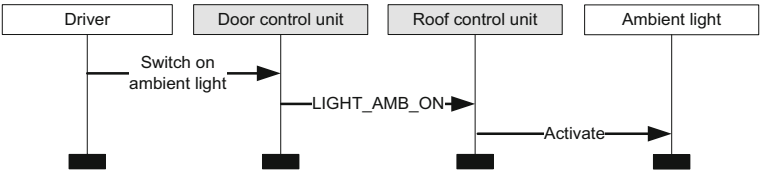


Fig. 5. Simple example of a component scenario

Note that one system scenario may be detailed by several component scenarios which define different possibilities for realising the system scenario through different system-internal interactions.

3.2.2 Definition of Additional Component Scenarios

Typically, additional component scenarios must be defined to deal with specific conditions that are considered at the component level such as temporary or permanent component failures or error diagnosis functionality (see Fig. 6 for an example). These scenarios may or may not include interactions with external actors.

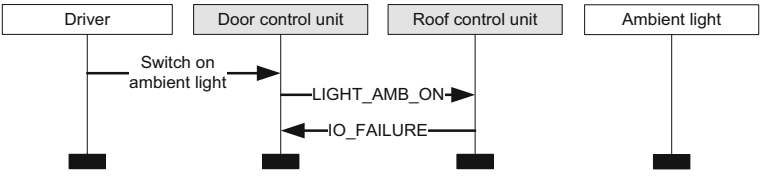


Fig. 6. Example of an additional, level-specific scenario

3.2.3 Component-Level Use Cases

Similar to the grouping of system scenarios into system-level use cases, component scenarios are also grouped into component-level use cases. A necessary condition for the completeness of the component-level use cases is that each system-level use case is detailed by at least one component-level use case. However, our approach does not require that the HMSC structure of the component-level use case is identical with the HMSC structure of the corresponding system-level use case (see Fig. 7). Rather, we advise that the component-level use cases are structured in a way that is convenient for the component level. For instance, it may be convenient to decompose a component-level use case into several more fine-grained use cases and relate the composed use case to the system-level use case. Furthermore, the comprehensibility and changeability of component-level use cases can often be improved by extracting and merging redundant scenario fragments.

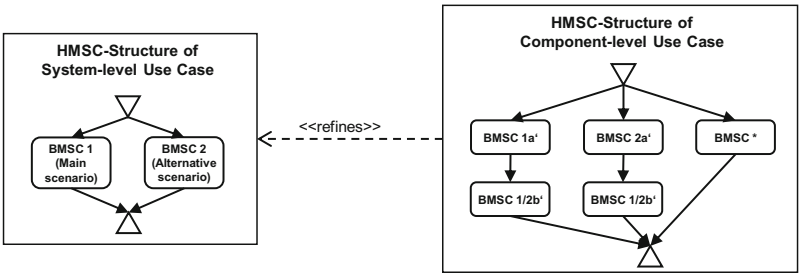


Fig. 7. HMSC structures of a system-level and a component-level use case

3.3 Comparison of System and Component Scenarios

The goal of comparing system and component scenarios is to identify differences in the specified sequences of interactions at the system level and the component level. The possible causes of such differences include:

- Omission of a specific system-actor interaction at some abstraction level
- Definition of an additional system-actor interaction at some abstraction level
- Changed order of the interactions at the system or the component level
- Definition of an additional scenario at some abstraction level that causes a difference in the specified, external behaviour

The comparison is performed for each pair of scenarios related by a "refines" link (see Fig. 7). The comparison can be applied to individual scenarios (i.e. one system scenario is compared to one component scenario), to use cases (i.e. a system-level use case is compared to a component-level use case), or to sets of interrelated use cases (see Section 2.2). The decision whether a detected difference is considered as an inconsistency or not is influenced by project-specific consistency rules. These rules can, for instance, allow or forbid the definition of additional system-actor interactions at the component level (see Section 4.4).

In case an inconsistency is identified, human judgement is required to determine necessary corrections. For instance, an additional sequence of interactions at the component level may be caused either by a missing system scenario or by an unwanted, additional component scenario. Hence, the stakeholders must decide whether the system scenarios, the component scenarios, or even both must be corrected to resolve a detected inconsistency.

4 Computation of Differences between Scenarios across Two Abstraction Levels

In this section, we outline our algorithm for computing the differences between scenarios defined at two different abstraction levels. Fig. 8 shows an overview of the major steps of the algorithm. The input of the algorithm comprises two message sequence chart (MSC) specifications, i.e. a system-level and a component-level specification, and, in addition, the coarse-grained system architecture. The MSC

specifications represent a pair of scenarios (or use cases) related to each other by a "refines" link. Each MSC specification may comprise several HMSCs and BMSCs. The algorithm is applied to each such pair of MSC specifications individually. The architecture that is provided as input defines the decomposition of the system into a set of components. It hence interrelates the instances defined in the two MSC specifications.

In Step 3.1 (see Fig. 8), the scenarios are normalised in order to ensure that identical message labels have identical meanings. Step 3.2 transforms the normalised MSCs into interface automata. It results in a system-level automaton P_H (Step 3.2a) and a component-level automaton P_L (Step 3.2b). Step 3.3 computes the differences between P_H and P_L. It results in two automata P_{H-L} and P_{L-H}.

The individual steps are explained in more detail in Subsections 4.1 to 4.3. In addition, we outline the analysis of the computed difference automata in Section 4.4.

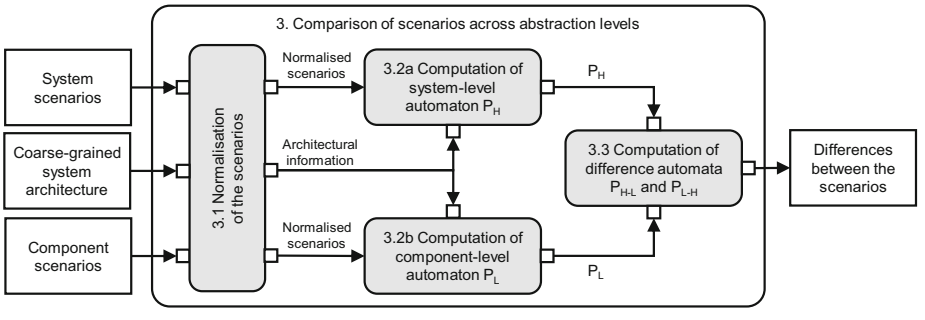


Fig. 8. Main steps of the comparison of system and component scenarios

4.1 Normalisation of the Message Sequence Charts

To compute the differences between the system and component scenarios, the message sequence charts documenting the scenarios must match some input criteria. We assume that each message sent or received by an instance has a label that uniquely identifies its message type. Furthermore, we assume that messages of a specific type sent to the environment or received from the environment are labelled identically at the system level and the component level.

The information which components decompose the system is taken from the architecture model. For the system scenarios, we assume that the instance representing the system is named consistently with the representation of the system in the architecture model. For the component scenarios, we assume that the instances representing system components are named consistently with the representations of the components in the architecture model.

To ensure that the above assumptions hold, a normalisation step is performed in which the instance names and message labels are checked and adapted, if necessary.

4.2 Transformation of the Scenarios into Interface Automata

To facilitate computing the differences between system and component scenarios, we employ a transformation of MSCs into automata. Interface automata [3] offer several

advantages that make them particularly suitable for our approach. For instance, the set of actions of an interface automaton is partitioned into a set of input actions, a set of output actions, and a set of internal actions. This corresponds to the event types defined for MSCs (receive, send, internal). Furthermore, interface automata do not enforce that each input action is enabled in every state. This is also true for the MSCs in our approach since we assume that, after reaching a specific location of an instance line, only the specified events are allowed to occur.

We briefly outline the transformation of the scenarios into interface automata:

1. *Construction of instance automata*: In this step, an interface automaton is computed for the system (system level) as well as for each component (component level). For this purpose, first each BMSC is transformed into a set of partial automata based on the algorithm described e.g. in [10]. Subsequently, the partial automata are concatenated by inserting τ-transitions (i.e. invisible transitions) as defined by the HMSC edges. Environment instances are disregarded in this step.
2. *Elimination of τ-transitions and indeterminism*: In this step, non-deterministic transitions and the τ-transitions inserted during concatenation are eliminated in the interface automata that were constructed in the first step. For performing this step, standard algorithms for automata such as those described in [11] can be used. The results of this step are the system-level automaton P_H and a set of component-level automata.
3. *Composition of the instance automata*: In this step, the automata derived from the component scenarios are composed to a single component-level automaton P_L by means of the composition operator defined for interface automata (see [3]).

4.3 Computation of the Differences between the Scenarios

The comparison of the automata shall reveal differences between the traces of the automata with regard to the externally observable system behaviour (similar to weak trace equivalence; see [12]). For this purpose, the traces consisting only of input and output actions of the interface automata P_H and P_L need to be compared with each other. The set of traces of the automaton P_H is called the language of P_H and denoted as L_H. The set of traces (of input and output actions) of P_L is called the language of P_L and denoted as L_L. To compare the two languages, two differences must be computed:

- $L_{H\text{-}L} = L_H \setminus L_L = L_H \cap \neg L_L$ and
- $L_{L\text{-}H} = L_L \setminus L_H = L_L \cap \neg L_H$

Hence, for computing the desired differences, the intersection and the complement operator for automata must be applied [11]. Since the complement operator requires a deterministic finite automaton as input, P_H and P_L must be transformed into deterministic automata. Furthermore, the internal actions defined in P_L must be substituted by τ-transitions. Due to space limitations, we omit the details here.

4.4 Analysis of the Computed Differences

The requirements engineers can interpret the resulting automata in the following way:

- $L_{H\text{-}L} = \emptyset$ and $L_{L\text{-}H} = \emptyset$: In this case the externally observable traces of both automata are identical, i.e. the scenarios at the system level and the component are consistent to each other.
- $L_{L\text{-}H} \neq \emptyset$: In this case, the component scenarios contain traces that are not defined at the system layer. The requirements engineers have to analyse these in order to determine which of them are valid or desired and which ones are not. Traces contained in $L_{L\text{-}H}$ that are considered valid may indicate missing system scenarios. However, the project-specific consistency rules may also allow such traces under certain conditions.
- $L_{H\text{-}L} \neq \emptyset$: In this case, the system scenarios contain traces that are not defined at the component level. The requirements engineers have to analyse these traces in order to determine which of these traces are valid. Traces contained in $L_{H\text{-}L}$ that are considered valid indicate missing component scenarios.

For supporting the interpretation and analysis of the computed differences, we generate graphical representations of the difference automata using the environment described in [13]. The analysis results are used to drive the further development and consolidation of the system and component scenarios.

5 Evaluation of the Approach

We have performed a preliminary evaluation of our approach by applying it to a (simplified) adaptive cruise control (ACC) system based on [14]. In a first step, use cases were identified for the ACC system, both, at the system level and the component level. All in all, eleven use cases were identified. Fig. 9 shows an excerpt of the use case diagram.

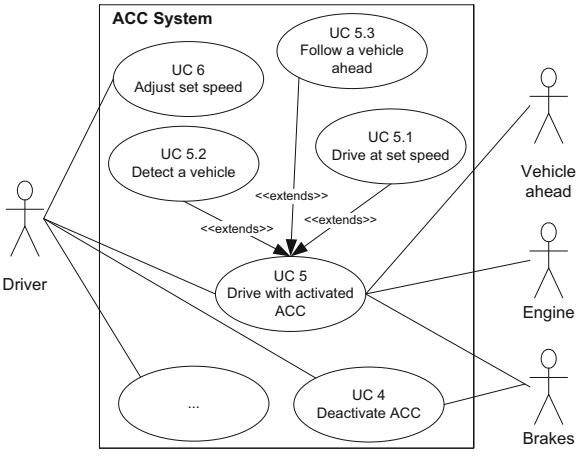


Fig. 9. Excerpt of the use case diagram defined for the ACC system

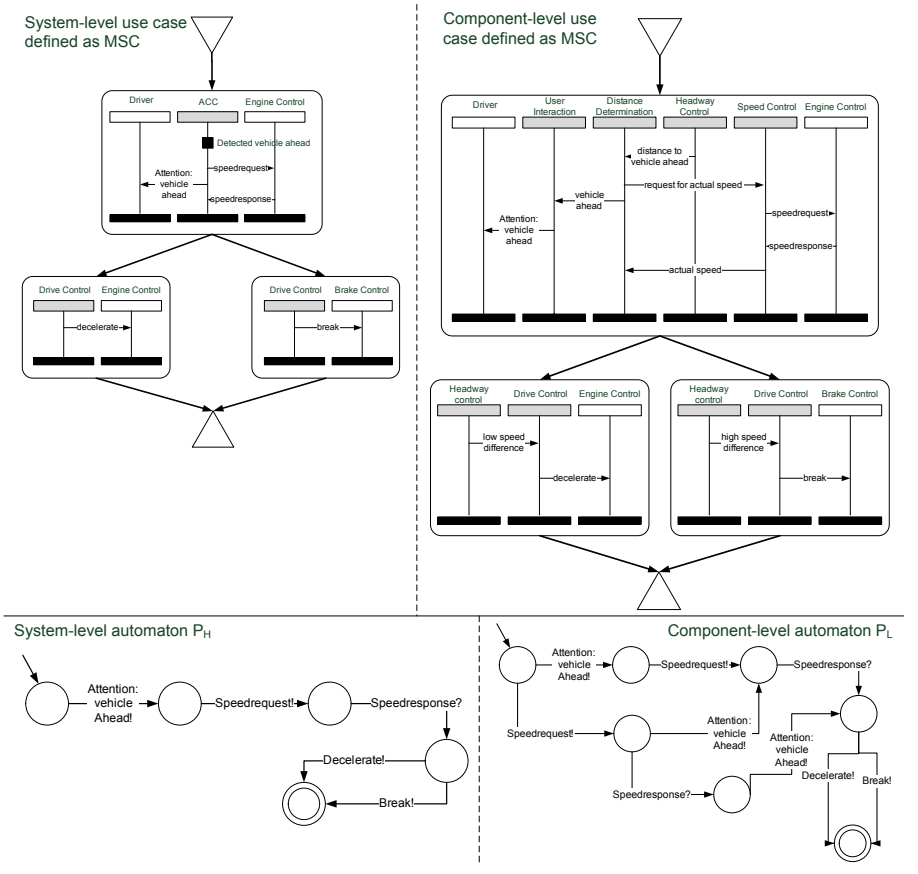


Fig. 10. System- and component-level specifications of an exemplary use case

For the identified use cases, scenarios were specified by means of message sequence charts. The specification activity resulted in eleven HMSCs and nineteen BMSCs at the system level and eleven HMSCs and twenty-five BMSCs at the component level. The consistency checking was performed using a prototypical implementation of the algorithm described in Section 4. Based on the computed differences, the system scenarios and component scenarios were consolidated to remove inconsistencies. Fig. 10 depicts an exemplary system-level use case defined for the ACC system, the corresponding component-level use case as well as the automata P_H and P_L computed for these use cases. The difference $L_{H\text{-}L}$ for the use case depicted in Fig. 10 is empty which means that the component-level use case realises all scenarios defined by the system-level use case. The automaton representing the difference $L_{L\text{-}H}$ is shown in Fig. 11. The sequences of actions which lead from the start state to the final state of this automaton are included in the component-level use case but are not defined by the system-level use case and thus may indicate inconsistencies. The evaluation demonstrated the importance of an automated, tool-supported consistency

check between system and component scenarios. The simplified ACC system used in the evaluation was already too complex to detect all inconsistencies between system and component scenarios manually. The prototypical tool revealed a large number of inconsistencies in the initial use cases and thus contributed significantly to improving the consistency of the use cases across the two abstraction levels. However, since the evaluation has been performed in an academic environment, further investigations concerning the applicability and usefulness of the approach in an industrial environment are needed.

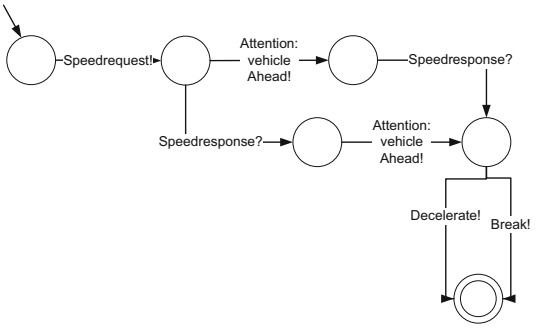


Fig. 11. Difference automaton representing $L_{L\text{-}H}$

Scalability or performance problems did not occur during the consistency check of the ACC scenarios. For instance, the computation of the difference automaton depicted in Fig. 11 took approximately 100 milliseconds. Still, for very complex use cases (such as a composition of all use cases defined for a system into a single use case), a more efficient implementation might be needed.

6 Related Work

Although scenario-based approaches exist that support, in principle, the development of scenarios at different abstraction levels such as FRED [15] and Play-in/Play-out [16], these approaches do not offer the required rigor for safety-oriented development. The checking of the consistency of the scenarios across abstraction levels is not supported by these approaches. Play-in/Play-out merely supports checking whether a set of "universal" scenarios realise a defined "existential" scenario which is not sufficient for proving cross-level consistency.

Approaches that support the formal verification of scenarios suffer from other deficiencies limiting their applicability within our approach. Existing techniques that, in principle, support the verification of MSCs across different abstraction levels provide insufficient support for HMSCs or do not support HMSCs at all. In [9], severe restrictions are imposed by requiring identical HMSC structures at the system level and the component level. The goal of temporal-logic model checking is typically to reveal a single counter example. In contrast, our approach computes extensive differences

between the scenarios defined at two abstraction levels. Furthermore, to apply temporal-logic model checking, use cases must be encoded using temporal logic which limits the applicability of such an approach in practice.

Furthermore, our approach can be regarded as a further step towards a methodical support for the transition between requirements and design as it facilitates the formally consistent specification of black-box scenarios and design-level scenarios. The approach thus complements less formal approaches such as [18] which aim at supporting the communication between requirements engineers and architects.

7 Conclusion

The approach presented in this paper closes a gap in the existing, scenario-based requirements engineering methods. It supports the development of scenarios at different abstraction levels and therein facilitates cross-level consistency checking. Cross-level consistency is important, for instance, for constructing safety proofs and to avoid requirements defects which lead to significant rework during system integration.

The approach employs the message sequence charts (MSC) language as a formal, visual specification language for scenarios and use cases. Individual scenarios are specified as basic message sequence charts (BMSCs). High-level message sequence charts (HMSCs) interrelate several BMSCs and allow for iterations and alternatives. The consistency check offered by our approach aims at detecting differences in the traces of externally observable events specified at the system level and those specified at the component level. The approach thus reveals, for instance, whether the traces of a component-level MSC are complete and necessary with respect to a system-level MSC. The approach is not based on simple, syntactic correspondences but rather employs a transformation of MSCs into interface automata. This makes the approach robust against changes at the syntactic level such as restructuring an MSC.

We have demonstrated the feasibility of our approach by applying it to the specification and consistency checking of requirements for an adaptive cruise control system. Our approach has proven useful for supporting the specification of the scenarios at the system and component level. We hence consider objectives O1 and O2 defined in Section 1.3 to be met. The consistency of the scenarios was checked using a prototypical tool. Thereby, a large amount of inconsistencies could be resolved which were difficult or even impossible to detect manually. We hence consider objective O3 (see Section 1.3) to be met. The approach can be applied in settings where consistency across different abstraction levels must be enforced and the use of formal specification and verification methods is accepted. A detailed evaluation of the applicability of our approach in industrial settings is ongoing work.

Acknowledgements. This paper was partly funded by the German Federal Ministry of Education and Research (BMBF) through the project “Software Platform Embedded Systems (SPES 2020)”, grant no. 01 IS 08045. We thank Nelufar Ulfat-Bunyadi for the rigorous proof-reading of the paper.

References

[1] Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering Journal (REJ) 11, 79–101 (2006)

[2] International Telecommunication Union. Recommendation Z.120 - Message Sequence Charts, MSC (2004)

[3] De Alfaro, L., Henzinger, T.A.: Interface Automata. In: Proc. of the ACM SIGSOFT Symp. on the Foundations of Software Engineering, pp. 109–120 (2001)

[4] RTCA: DO-178B – Software Considerations in Airborne Systems and Equipment Certification (1992)

[5] Potts, C.: Using Schematic Scenarios to Understand User Needs. In: Proc. of the ACM Symposium on Designing Interactive Systems – Processes, Practices, Methods and Techniques (DIS 1995), pp. 247–266. ACM, New York (1995)

[6] Pohl, K.: Requirements Engineering – Foundations, Principles, Techniques. Springer, Heidelberg (to appear 2010)

[7] Peled, D.: Specification and Verification using Message Sequence Charts. Electr. Notes Theor. Comp. Sci. 65(7), 51–64 (2002)

[8] Whittle, J., Schumann, J.: Generating Statechart Designs from Scenarios. In: Proc. of the Intl. Conference on Software Engineering, pp. 314–323 (2000)

[9] Khendek, F., Bourduas, S., Vincent, D.: Stepwise Design with Message Sequence Charts. In: Proc. of the IFIP TC6/WG6.1, 21st Intl. Conference on Formal Techniques for Networked and Distributed Systems, pp. 19–34. Kluwer, Dordrecht (2001)

[10] Krüger, I., Grosu, R., Scholz, P., Broy, M.: From MSCs to Statecharts. In: Proc. of the IFIP WG10.3/WG10.5, Intl. Workshop on Distributed and Parallel Embedded Systems, pp. 61–71. Kluwer, Dordrecht (1999)

[11] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley, Reading (2006)

[12] Milner, R.: Communication and Mobile Systems – The Pi Calculus. Cambridge University Press, Cambridge (1999)

[13] Gansner, E., North, S.: An Open Graph Visualization System and its Applications to Software Eengineering. Software - Practice and Experience (1999)

[14] Robert Bosch GmbH: ACC Adaptive Cruise Control. The Bosch Yellow Jackets (2003), http://www.christiani-tvet.com/

[15] Regnell, B., Davidson, A.: From Requirements to Design with Use Cases. In: Proc. 3rd Intl. Workshop on Requirements Engineering – Foundation for Software Quality, Barcelona (1997)

[16] Harel, D., Marelly, R.: Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Heidelberg (2003)

[17] Ohlhoff, C.: Consistent Refinement of Sequence Diagrams in the UML 2.0. Christian Albrechts Universität, Kiel (2006)

[18] Fricker, S., Gorscheck, T., Byman, C., Schmidle, A.: Handshaking with Impementation Proposals: Negotiating Requirements Understanding. IEEE Software 27(2), 72–80 (2010)

Requirements Value Chains: Stakeholder Management and Requirements Engineering in Software Ecosystems

Samuel Fricker

University of Zurich, Department of Informatics
Binzmuehlestrasse 14, 8057 Zurich, Switzerland
fricker@ifi.uzh.ch

Abstract. [Context & motivation] Market-oriented development involves the collaboration of many stakeholders that do not necessarily directly interact with a given development project but still influence its results. These stakeholders are part of the requirements value chain for the concerned software product. **[Question/problem]** Understanding the structure and functioning of requirements value chains is essential for effective stakeholder management and requirements engineering within the software product's ecosystem. **[Principal ideas/results]** The paper explores and exemplifies fundamental concepts that are needed to characterize and reason about requirements value chains. **[Contribution]** This characterization is used to describe the relevant knowledge landscape and to suggest research avenues for understanding the principles needed for managing requirements-based stakeholder collaboration.

Keywords: Requirements value chain; Software ecosystems; Requirements negotiation; Vision.

1 Introduction

Much requirements engineering research has focused on engineering requirements of a system with few easily accessible stakeholders [1]. This model of stakeholder involvement is adequate for many bespoke situations, but is too simplistic for market-driven software development [2, 3], where collaboration among stakeholders is a central concern [4]. Here, a possibly large number of anonymous and distributed users of current product versions provide feedback from product use and state new requirements. Developers state requirements that emerge from attempts to balance customer satisfaction and development feasibility. Marketing and management departments define development scope. Other roles pursue further objectives.

Stakeholders and their relationships are a central concern of software ecosystems, where stakeholder collaboration is organized in two value chains [5]. The *requirements value chain* applies to inception, elaboration, and planning of software, starting with business and application ideas and ending with an agreed detailed set of requirements for implementation. It is concerned of discovering, communicating, and matching interests of direct and indirect stakeholders with functionality and quality properties of the software to be developed [6]. Stakeholders need to be known and differentiated [7, 8] and conflicting perspectives and goals resolved [9, 10].

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 60–66, 2010.

Concluded development of the software allows transiting from the requirements value chain to the *supply chain* [5]. The supply chain applies to the execution phase in the software lifecycle. It is concerned of the production, distribution, selection, composition, and operation of the software to make it available to end users [11-13]. Hence, the supply chain is concerned of satisfying the needs discovered and aligned in the requirements value chain by delivering the results from development and providing services based thereon in a profitable and sustainable manner [14].

The requirements value chain is little understood so far beyond project stakeholder management and goal modeling. It is unclear which requirements communication, collaboration, and decision-making principles lead to efficient, value-creating and sustainable alignment of interests between interdependent stakeholders across software projects and products. Industry cases from large-scale process development [15, 16], inter-company collaboration [17], and global software engineering [18, 19] show that bringing transparency into the requirements value chain is important. Proper stakeholder collaboration leads to accepted products and innovation [20].

This paper uses fundamental concepts from negotiation [21] to explore these facets of requirements value chains. The result provides a basis to reflect on research needed for an improved understanding of value creation and of collaboration, hence for better managing the political and strategic context of requirements engineering [22].

The paper is structured as follows. Section 2 explores requirements value chains from structural and dynamic perspectives and exemplifies. Section 3 discusses emerging research issues. Section 4 summarizes and concludes.

2 Requirements Value Chains

2.1 A Negotiation-Based View of Requirements Value Chains

Negotiation is an inherent part of decision-making between stakeholders [21]. This view emphasizes the social and political aspects of requirements engineering and assumes interest in collaboration and value creation. It has successfully been applied in project requirements engineering [23]. Value-creating negotiations require proposal and exploration of alternatives for matching stakeholders' interests to find win-win agreements worth more in total than if each party would act on its own.

A win-win negotiation process can be characterized as follows [21]. Two or more *stakeholders* get in contact with each other, share their *positions* in terms of interests and expectations, seek alternatives, and get to an *agreement*. Negotiations can be about *alignment of interests* or *sharing of scarce resources*. Negotiations are *small-scale* if they just affect the negotiators, or *large-scale* if they involve a myriad of players and issues in the so-called secondary, hidden negotiation table.

Social Structure: Stakeholders and Relationships. Social structure relates to stakeholders, someone or a group of individuals who gains or loses something as a result of a software project [7], and their relationships. Stakeholders embody direct or indirect viewpoints towards the software [8]. Direct viewpoints concern software operation and include the product's user. Indirect viewpoints influence software success indirectly and include development execution, financing, and regulation.

Orderer-contractor, group membership, and delegation relationships connect stakeholders. Orderer-contractor relationships connect two layers in the requirements value chain with each other [24]. Group membership groups stakeholders with similar interests, same goals, or interdependencies [25]. Delegation relationships point from a principal stakeholder to an agent [21]. Agents are employed when they have more negotiation expertise than the primary stakeholder, deeper domain knowledge, or a network and special influence that enables rapid achievement of effective agreements.

A central stakeholder in the requirements value chain of a software product is the *product manager* [3] that coordinates large-scale negotiations by performing requirements management, roadmapping, and release planning. Concerned *company-internal stakeholders* include senior managers that supervise the organization, marketing and sales that interface to customers, development that implements the product, production that makes the product available to the supply chain, service and support that enable effective use of the product, and controlling that measures product performance. *Company-external stakeholders* include market intelligence and those of the supply chain: users, customers, channels, suppliers, and competitors.

Information Structure: Interests and Agreements. Interest are objectives pursued by stakeholders [21]. They are often related to the stakeholder's role in the ecosystem. Company management pursues business goals. Customers profit from benefits generated by the software, and users from functionality, reliability, usability, and efficiency the software. Development is interested in solution design and technology.

An *agreement* is a settling between stakeholders that results from successful negotiation and enables collaboration of the concerned stakeholders. Agreements are a form of conflict resolution. They describe how selected interests of the negotiation partners contribute to each other when specifying *interest alignment*, and plans for resource use when specifying *allocation of scarce resources*.

Documents and *data stores* are used to capture positions and agreements in product management. Positions of users are captured as needs and support cases, of customers as market requirements, of research and development as ideas, and of product management as product vision. Product requirements are used for triage and preparing negotiations with key stakeholders. Resulting agreements are documented with product strategies, roadmaps, and release plans. Development is contracted with implementation proposals and requirements specifications.

Dynamics: Requirements Value Chain Evolution. Requirements value chains evolve when stakeholders enter or leave the software ecosystem, specialize and become members of groups, and establish relationships. Active integration may involve application for a given role. Passive involvement happens through personal interests, traits and relationships that make the person attractive for being integrated.

People acquire *group membership* if they pursue the same goals as other group members, if they are interdependent with other members, if they interact with other members, or if they share a set of norms [26]. Group enrolment is active or passive. The constitution of groups influences negotiation tactics and methods [25].

Software product management establishes and maintains a software ecosystem by managing stakeholders and studying and aligning their interests. With requirements triage a product manager decides about relevance of stakeholders and interests.

Contracting involves the selection of development teams and suppliers. Budgeting and release planning further constrains stakeholder involvement.

2.2 Example of a Requirement Value Chain

Examples of requirements value chains have been published [15, 16, 18, 19, 25], mostly characterized ad-hoc. Figure 1 shows an interpretation of one of them using the notation introduced in [16]. It describes an institutionalized requirements value chain that was developed in a large-scale requirements engineering process development effort [15]. Process development defined the social network and requirements engineering artifacts by identifying roles (circles in Figure 1), their responsibilities with respect to the company's product and technology portfolio, and documentation to capture agreements between the roles (arrows in Figure 1). Process development left open the concrete interests to be pursued and aligned.

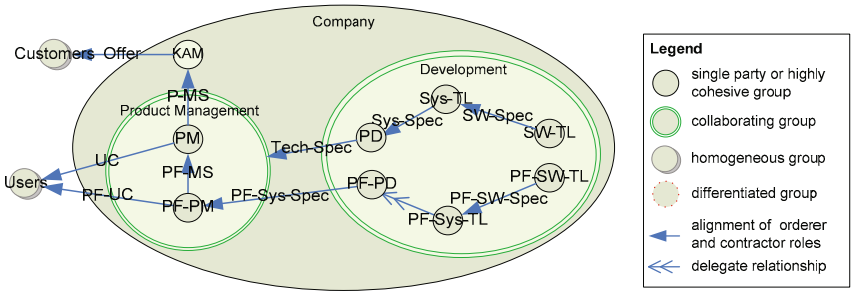


Fig. 1. Multi-project development case (abbreviations in [15])

Analysis of the requirements value chain in Figure 1 raises questions regarding efficiency and completeness of the developed process. Software usability, if a concern for the company's products, requires alignment of user interests with the software team leader's (SW-TL) intentions over four specifications. The effort needed could be significantly reduced and the quality of the alignment improved by introducing more direct collaboration with selected users. The requirements value chain excludes relationships to production and suppliers, which could connect development to the supply chain and would allow considering requirements that affect product cost, hence a substantial part of the economic success. Company resources, finally are considered mere implementation resources. No link points from product management to development that would allow gathering innovative ideas [20].

3 Research Issues

Conceptualizing software product stakeholders as a requirements value chain can bring transparency into how the software ecosystem affects product inception. Research in this area can provide the fundaments for reasoning on efficiency and effectiveness of requirements engineering strategy and of innovation.

Requirements value chain analysis can allow understanding power of given stakeholders, process performance, and ecosystem maturity. *Social network theory* [27] provides concepts and models for determining stakeholder power and influence and for evaluating structural strengths and weaknesses of the stakeholder network. *Group theory* [26] gives insights into group effectiveness and the development of specialized roles. *Negotiation theory* [21] provides decision-making knowledge.

Stakeholders need to be managed in the software ecosystem to evolve a value chain. Partners need to be identified, relationships established, stakeholder interactions monitored, and changes in requirements value chain controlled. *Partner identification* may be addressed by directories or by recommendation systems. Established social networks provide such capabilities, but have not been used for such purposes in requirements engineering yet. *Groups*, besides negotiation tactic selection [25], can also play a role for partner and peer identification. *Group management* addresses group lifecycle, performance, and partnering with other groups. *Relationships need to be established* to allow partners to start negotiations. Factors like trust and distance affect the quality of such relationships. *Value chain management*, opposed to passive emergence of a chain, involves proactive value chain composition, structuring, and change to provide value and perspectives to its members and to ensure sustainability of the software ecosystem.

Information spread in the requirements value chain needs to be managed. The choice of interest elicitation, expectation setting, and decision documentation approaches can have effects on the transparency and performance of the value chain. *Computer-supported collaborative work* [28], traceability, and audit trails can contribute to understanding effective information sharing and management. *Social network technologies* may give unprecedented support for requirements engineering.

Requirements value chain structure can affect innovation, requirements engineering performance and software success. Negotiations permit local alignment of interest, but may not be effective for global alignment. Distance, feed-forward and feedback affect the overall alignment of stakeholder interests and intentions in the value chain and the motivation of stakeholders to collaborate. A new management role may be needed, responsible for value chain structure and policies, for guiding stakeholder behavior, and for controlling progress and success of interest alignment.

4 Summary and Conclusions

This paper proposed a fresh view on stakeholder involvement in requirements engineering. It has introduced and exemplified the concept of requirements value chains where requirements emerge from and propagate with inter-stakeholder collaboration. The resulting view on stakeholder management and requirements engineering, which we recommend to address with negotiation principles, can provide insights for managing the political and strategic aspects of requirements engineering beyond the horizon of a development project. Characteristic for a vision on our field, a lot of research remains for illuminating our understanding.

References

1. Cheng, B., Atlee, J.: Research Directions in Requirements Engineering. In: Future of Software Engineering (FOSE 2007). IEEE Computer Society, Los Alamitos (2007)
2. Regnell, B., Brinkkemper, S.: Market-Driven Requirements Engineering for Software Products. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 287–308. Springer, Heidelberg (2005)
3. Ebert, C.: Software Product Management. Crosstalk 22, 15–19 (2009)
4. Karlsson, L., Dahlstedt, Å., Regnell, B., Natt Och Dag, J., Persson, A.: Requirements engineering challenges in market-driven software development - An interview study with practitioners. Information and Software Technology 49, 588–604 (2007)
5. Messerschmitt, D., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. The MIT Press, London (2003)
6. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: IEEE Intl. Symp. on Requirements Engineering, Annapolis MD, USA (1997)
7. Alexander, I., Robertson, S.: Understanding Project Sociology by Modeling Stakeholders. IEEE Software 21, 23–27 (2004)
8. Kotonya, G., Sommerville, I.: Requirements Engineering with Viewpoints. Software Engineering Journal 11, 5–18 (1996)
9. van Lamsweerde, A., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Software Engineering 24, 908–926 (1998)
10. Easterbrook, S., Nuseibeh, B.: Using ViewPoints for Inconsistency Management. Software Engineering Journal 11, 31–43 (1996)
11. Jansen, S., Brinkkemper, S., Finkelstein, A.: Providing Transparency in the Business of Software: A Modeling Technique for Software Supply Networks. Virtual Enterprises and Collaborative Networks (2007)
12. Lauesen, S.: COTS Tenders and Integration Requirements. Requirements Engineering 11, 111–122 (2006)
13. Rayport, J., Sviokla, J.: Exploiting the Virtual Value Chain. Harvard Business Review 73, 75–85 (1995)
14. Gordijn, J., Yu, E., van der Raadt, B.: e-Service Design Using i* and e3value Modeling. IEEE Software 23, 26–33 (2006)
15. Paech, B., Dörr, J., Koehler, M.: Improving Requirements Engineering Communication in Multiproject Environments. IEEE Software 22, 40–47 (2005)
16. Fricker, S.: Specification and Analysis of Requirements Negotiation Strategy in Software Ecosystems. In: Intl. Workshop on Software Ecosystems, Falls Church, VA, USA (2009)
17. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT Support for Release Management Processes in the Automotive Industry. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 368–377. Springer, Heidelberg (2006)
18. Damian, D., Zowghi, D.: RE Challenges in Multi-Site Software Development Organisations. Requirements Engineering 8, 149–160 (2003)
19. Damian, D.: Stakeholders in Global Requirements Engineering: Lessons Learned from Practice. IEEE Software 24, 21–27 (2007)
20. Gorschek, T., Fricker, S., Palm, K., Kunsman, S.: A Lightweight Innovation Process for Software-Intensive Product Development. IEEE Software (2010)
21. Thompson, L.: The Mind and Heart of the Negotiator. Prentice-Hall, Englewood Cliffs (2004)

22. Bergman, M., King, J.L., Kyytinen, K.: Large-Scale Requirements Analysis Revisited: The Need for Understanding the Political Ecology of Requirements Engineering. Requirements Engineering 7, 152–171 (2002)
23. Grünbacher, P., Seyff, N.: Requirements Negotiation. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 143–162. Springer, Heidelberg (2005)
24. Fricker, S., Gorschek, T., Byman, C., Schmidle, A.: Handshaking with Implementation Proposals: Negotiating Requirements Understanding. IEEE Software 27, 72–80 (2010)
25. Fricker, S., Grünbacher, P.: Negotiation Constellations - Method Selection Framework for Requirements Negotiation. In: Working Conference on Requirements Engineering: Foundation for Software Quality, Montpellier, France (2008)
26. Johnson, D., Johnson, F.: Joining Together: Group Theory and Group Skills. Pearson, London (2009)
27. Wasserman, S., Faust, K.: Social Network Analysis, Cambridge (2009)
28. Gross, T., Koch, M.: Computer-Supported Cooperative Work. Oldenbourg (2007)

Binary Priority List for Prioritizing Software Requirements

Thomas Bebensee, Inge van de Weerd, and Sjaak Brinkkemper

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands
{tbebense,i.vandeweerd,s.brinkkemper}@cs.uu.nl

Abstract. [Context and motivation] Product managers in software companies are confronted with a continuous stream of incoming requirements. Due to limited resources they have to make a selection of those that can be implemented. However, few prioritization techniques are suitable for prioritizing larger numbers of requirements. [Question/problem] Binary Priority List (BPL) is a binary search based technique for prioritizing requirements. Academics and practitioners have referred to it in previous works. However, it has not been described and researched in detail. [Principal ideas/results] This paper introduces BPL, examines how it can be used for prioritizing requirements and assesses its prioritization process quality by comparing it to another prioritization technique. A facilitating tool was developed and applied in two small Dutch product software companies. [Contribution] The paper demonstrates that the technique can be successfully used to prioritize requirements and is especially suitable for a medium amount of low-level requirements.

Keywords: Binary Priority List, Binary Search Algorithm, Requirements Prioritization, Software Product Management, Agile Project Management.

1 Introduction

Product software companies produce packaged software products aimed at a specific market [1]. In product software companies, the number of requirements from the market typically exceeds the number of features that can be implemented in one release due to limited resources. Requirements prioritization is aimed at responding to this challenge. It is defined as "an activity during which the most important requirements for the system (or release) should be identified" [2]. According to the reference framework for software product management, in which key processes, actors, and relations between them are modeled, it is the first step in the release planning process, "the process through which software is made available to, and obtained by, its users" [3]. The main actor in prioritization is the product manager, but other stakeholders (development, sales & marketing, customers, partners etc.) may influence it as well [3].

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 67–78, 2010.

One particular important stakeholder in software product management is the customer, or rather the representative of a large number of customers (market). Agile Project Management (APM) takes this into account by "energizing, empowering and enabling project teams to rapidly and reliably deliver customer value by engaging customer and continuously learning and adapting to their changing needs and environments" [4].

In the context of APM, product managers are responsible for requirements management, in contrast to the "traditional understanding" where it was the developers' responsibility [5]. The product manager is considered the customers' voice and in his role as the interface between the market and the development team. As such, he is also responsible for the prioritization of requirements [5]. In order to correspond to the prerequisite of an iterative (re)prioritization, agile prioritization techniques must reflect this dynamic nature. This enables delivering a maximized business value to the market throughout the project [5].

1.1 Requirements Prioritization Techniques

Prioritization of requirements is usually done during a prioritization session. Three stages in a prioritization session are distinguished [6]: (1) the preparation stage to structure the requirements and prepare the execution of the prioritization; (2) the execution stage where the actual prioritization is performed; and finally (3) the presentation stage where the prioritization results are presented.

Racheva *et al.* [5] review a number of agile requirements prioritization techniques. Based on the descriptions provided by them, these techniques can be classified into two main categories: techniques used to prioritize small amounts of requirements (small-scale) and techniques that scale up very well (medium-scale or large-scale), thus can be used for the prioritization of larger amounts of requirements (Racheva *et al.* talk about several dozen requirements).

Small-scale techniques can usually be used without the aid of a software tool and are often relatively simply structured. The techniques mentioned by Racheva *et al.* [5] are the round-the-group prioritization, the $100 allocation technique, the multi-voting system, the pair-wise analysis, the weighted criteria analysis, the dot voting technique, and the Quality Functional Deployment approach.

Medium-scale or large-scale techniques might be based on relatively complex algorithms or at least due to the large amount of requirements need tool support. In this category, Racheva *et al.* [5] refer to the MoSCoW technique, the Binary Priority List (they refer to it as "Binary Search Tree technique"), the Planning Game, and Wiegers's matrix approach (cf. [7]).

In addition, there are some techniques that do not fit very well into this scheme. One is ranking based on product definition, which could be used as a complement to other techniques to first derive a de-personalized measure of priority. Another technique is the application of mathematical programming techniques to interlink the whole release planning process. And yet another one is the Analytic Hierarchy Process, which is rather complex but limited to a small number of requirements due to the high number of comparisons necessary (cf. [8]). This technique has received considerable regard in literature (cf. [8], [6], [9]).

1.2 Research Question

Since product managers in product software companies usually have to prioritize larger amounts of requirements, medium or large-scale techniques are of special interest to them. In this paper we will therefore discuss how one particular of these techniques, which we call Binary Priority List (BPL), can be applied by product managers of such companies. Since it is a relatively simple technique, we expect it to be especially interesting for smaller product software companies that want to formalize their requirements prioritization process. The research question we want to answer is:

How can BPL be applied as a requirement prioritization technique in small product software companies and how reliable are its results?

In order to answer the research question, the remainder of the paper is structured as follows. Section 2 gives a rationale for the research done and explains the research approach used. In section 3, a detailed description of the BPL and the supporting tool is provided. Section 4 presents the case study approach and a description of the research sites and the results of the case studies are presented. Finally, in section 5, conclusions are drawn and areas of further research are indicated.

2 Rationale and Research Approach

Binary search is a popular algorithm to sort and search information [10]. A quick search on the Internet reveals a sizeable amount of publications on it (cf. [11], [12], [13]). However, this algorithm can also be used to prioritize software requirements. In this context we refer to it as Binary Priority List (BPL). There is only little notion of BPL in literature. Only two papers deal with it in more detail.

Karlsson *et al.* [6] compared BPL to five other prioritization techniques. In their research, BPL scored relatively weak in terms of time consumption, ease of use, reliability and fault tolerance. On the other hand, Ahl [8] conducted an experiment in which he compared BPL with four other prioritization techniques in terms of reliability of results, ease of use, time consumption and scalability. In his experiment BPL was considered the best out of the five techniques. Ahl comes to the conclusion that BPL scales up very well and is therefore especially interesting for prioritizing larger amounts of requirements.

A literature research did not reveal any evidence that BPL has been described in detail. Apparently, it has not been examined in the specific context of product software companies yet. We think that this makes it an interesting research object and we therefore describe how it can be applied in that context.

Our further research approach was the following:

1. *Description and tool support:* We created a detailed description on how to apply BPL for prioritizing software requirements. In addition, we developed a tool for product managers to use BPL as a prioritization technique for their software requirements.
2. *Case studies:* In order to validate the technique's application, we conducted case studies at two product software companies. The goal of these case studies was to test whether BPL can be applied as a requirements prioritization technique and

how it is perceived by experience software product managers. In addition, the technique's prioritization process quality, expressed by the factors reliability, time consumption and ease of use, was evaluated by comparing it to Wieger's prioritization approach.

This research approach has been designed according to the guidelines proposed by Hevner, March, Park, and Ram [14]. Case studies (cf. [15]) are considered very suitable for industrial evaluations of techniques [16].

3 Binary Priority List

Binary Search Tree (BST) is a widely used algorithm to sort and search information but it can also be used to prioritize requirements [6]. Applied to the context of requirements prioritization we refer to it as Binary Priority List (BPL). A systematical structure of the technique is shown in Figure 1.

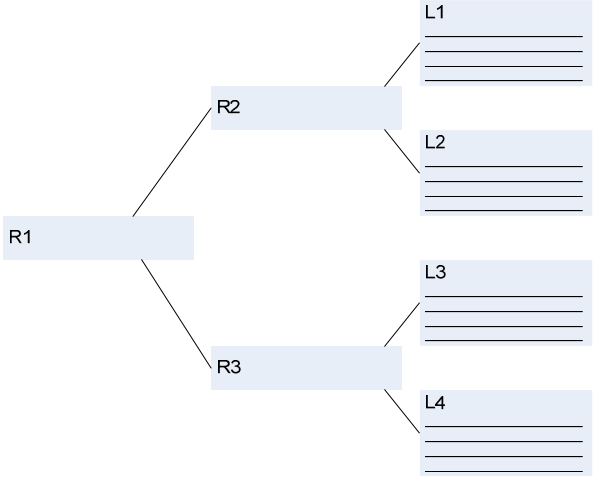


Fig. 1. Systematic structure of Binary Priority List

The figure shows a list of requirements containing three example requirements (R1, R2 and R3) and a number of sub-lists (L1, L2, L3 and L4) containing more requirements. In accordance with a binary tree structure (cf. [17] and [6]) requirements that are further up are more important than requirements further down. Therefore, R1's priority is lower than that of R2 but higher than that of R3. Subsequently, all requirements listed in the sub-lists L1 and L3 have higher and all requirements in L2 and L4 have a lower priority than their root requirement, R2 and R3 respectively.

The steps of applying the technique are (cf. [5], [6] and [8]):

1. Pile all requirements that have been collected from various sources.
2. Take one element from the pile, and use it as the root requirement.

3. Take another requirement and compare it in terms of priority to the root requirement.
4. If the requirement has a lower priority than the root requirements, compare it to the requirement below the root and so forth. If it has a higher priority than the root, compare it to the requirement above the root and so forth. This is done until the requirement can finally be placed as sub-requirement of a requirement without an appropriate sub-requirement.
5. Steps 2 to 4 are repeated for all requirements.
6. Finally, traverse the list from top to down to get the prioritized order of the requirements.

BPL can be applied by placing a number of cards, which represent the requirements, on a blackboard. However, when large numbers of requirements have to be processed, a software tool becomes necessary. Such a tool is also useful when the structure of the list is supposed to be altered in the future and should be stored electronically.

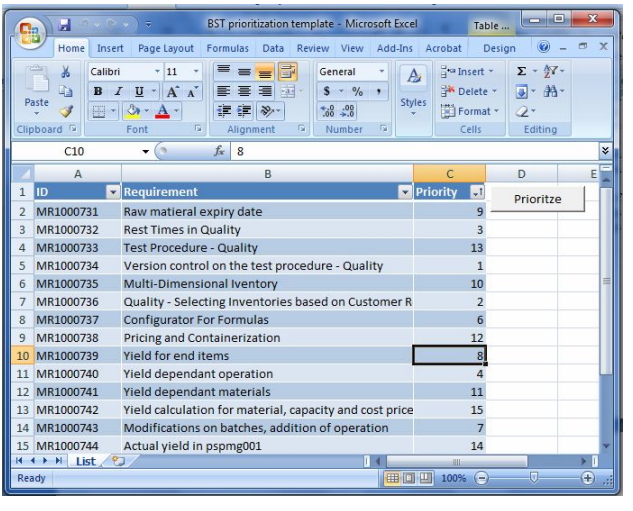


Fig. 2. Screenshot of the BPL tool

We developed a simple spreadsheet tool based on Microsoft Excel (see Figure 2). A macro guides the user through the prioritization process by asking him to compare different requirements and deciding which one is respectively more important than the other. The list structure is saved in an implicit form in a hidden spreadsheet, which allows saving it and changing it at a later time without having to run the whole prioritization again.

Due to its simple nature we expect BPL to be especially useful in environments where no formal prioritization techniques have yet been used to assist the product manager in prioritizing his criteria. This type of environment is mostly likely to be found in small product software companies that have recently grown in terms of requirements to be processed.

4 Case Studies

4.1 Approach

The case studies were divided into three phases: (1) a prioritization with BPL, (2) a prioritization with Wiegers's technique, and (3) an evaluation of the two techniques. To reduce the number of confounding factors as proposed by Wohlin and Wesslen [16], the same way of input, namely Excel spreadsheets, were used.

We conducted case studies in two small product software companies in which we compared BPL with Wiegers's technique in terms of the following three factors:

1. *Time consumption:* indicates the time necessary to prioritize a certain number of requirements.
2. *Ease of use:* describes how easy it is to use the examined prioritization technique assessed by the respective product manager.
3. *Subjective reliability of results:* indicates how reliable the result of the prioritization technique is in the opinion of an experienced product manager and thus how applicable the technique is to the respective company.

The ultimate goal was to show that BPL can be applied by product managers in small product software companies to systematize their requirement prioritization practices. In order give an additional indication of the technique's relative prioritization process quality, we compared it with Wieger's approach, a commonly used prioritization technique that is applied to similar numbers of requirements as BPL (cf. [7], [18] and [19]).

4.2 Research Sites

The first case study took place at Edmond Document Solutions (below referred to as Edmond), a small Dutch product software company. Edmond is a specialist in providing document processing solutions to document intensive organizations. The company employs 15 people whereof six are involved in software development. The software development process is based on Scrum (cf. [20]), an agile software development method.

The current release planning process takes place in two stages. High-level requirements are discussed once a year among the product manager, the operations manager and the sales director. The selection and order of requirements is defined in an informal discussion between the three. To gain a good understanding of the market, they visit exhibitions, read journals and communicate with major customers. The product manager estimates the required resources and makes sure the needed resources do not exceed the resources available.

In the second stage of the release planning process, the product manager derives low-level requirements from the high-level requirements defined in the first stage. To manage requirements together with tasks, bugs, improvements and impediments he uses JIRA (cf. [21]), a bug and issue tracking software. Prioritization of low level requirements takes place by comparing them in pairs with each other. In this process no formal technique is used. Subsequently, he assigns the requirements to Scrum sprints, periods of four weeks where developers work on a certain number of requirements.

The second case study took place at Credit Tools (below referred to as CT), a small Dutch product software company. CT produces encashment management software.

Five out of the company's 25 employees are software developers. In addition, there are two outsourced software developers. The company's development method is Rapid Application Development (cf. [22]).

Requirements are generated from customer requests, from information acquired through consultants and from ideas generated by the company's owners. JIRA is used to collect requirements and to communicate with the outsourcing developers. There is no formal process of requirements prioritization and not all requirements are systematically noted down.

4.3 Results

At Edmond, the product manager had prepared a list of 68 low-level requirements. This number of requirements was not chosen intentionally but corresponded to the number of requirements to be prioritized at that moment. To perform a prioritization based on the Wiegers's approach, he had also estimated for each requirement its relative benefit, its relative penalty, its relative costs, and its relative risk. At CT, 46 low-level requirements were prioritized instead. Again, this corresponded to the product manager's current requirements list.

In the first phase of each case study, the list of requirements was copied into the BPL spreadsheet tool. The product manager then went through the prioritization process, in which he was asked to perform a pair-wise comparison of the requirements' importance following the BST algorithm.

In the second phase, the requirements and the corresponding values estimated before were copied into a spreadsheet prepared for Wiegers prioritization. The spreadsheet automatically calculated relative priorities based on the estimates provided. Subsequently, the requirements were sorted by increasing priority.

In the evaluation phase, the product managers were asked for their opinion on the two compared prioritization techniques, especially with regard to ease of use and reliability. In addition, the time needed to perform a prioritization based on the two techniques was compared. The results are shown in Table 1.

Table 1. Comparison of the two techniques

	BPL		**Wiegers**	
	Edmond	*CT*	*Edmond*	*CT*
Ease of use	8/10	8/10	7/10	4/10
Subjective reliability	7/10	7/10	4/10	5/10
Time consumption	30 min	20 min	120 min	50-60 min

The product managers of both companies had a quite positive impression of BPL, which is reflected by their rating of the technique in terms of ease of use and reliability (see Table 1). One indicated that the ten most highly prioritized requirements corresponded exactly to his own manual prioritization. Lower priority requirements, however, partly differed from it. He supposed that this could be caused by accidentally giving wrong answers while going through the prioritization process and proposed to improve the user interface by including a possibility to correct a wrong choice. The second product manager indicated that the informal approach to prioritizing

requirements that they had used so far has many similarities with BPL. Therefore, he considered it as quite suitable for his company.

To compare the results of two techniques, Table 2 shows the ten requirements with the highest priority according to both techniques.

Table 2. The ten most highly prioritized requirements according to both techniques

	Edmond		**CT**	
Priority	*BPL*	*Wiegers*	*BPL*	*Wiegers*
1	5	54	18	19
2	2	23	35	43
3	28	21	25	11
4	27	61	14	18
5	3	63	24	29
6	6	12	16	42
7	23	28	11	3
8	12	45	42	22
9	16	50	33	24
10	7	53	2	13
Avg. Diff.	**14.75**		**8.54**	

Figure 2 (Edmond case) and Figure 3 (CT case) show the priorities of all requirements assigned by Wiegers's techniques plotted over the priorities assigned by BPL, which is also represented by the straight line. The stronger the two graphs in each figure differ, the bigger the difference between the priorities assigned by the two techniques.

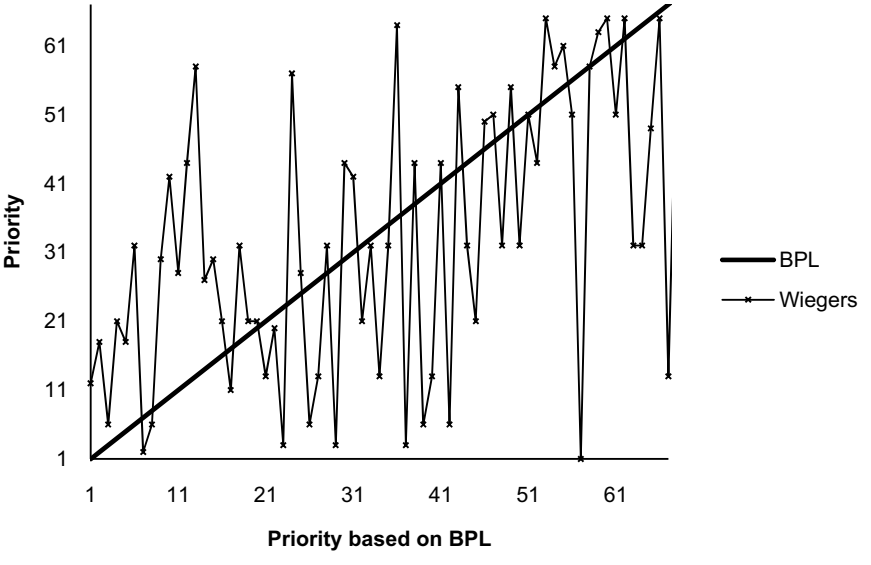


Fig. 3. Edmond Case: comparison of the two prioritization techniques

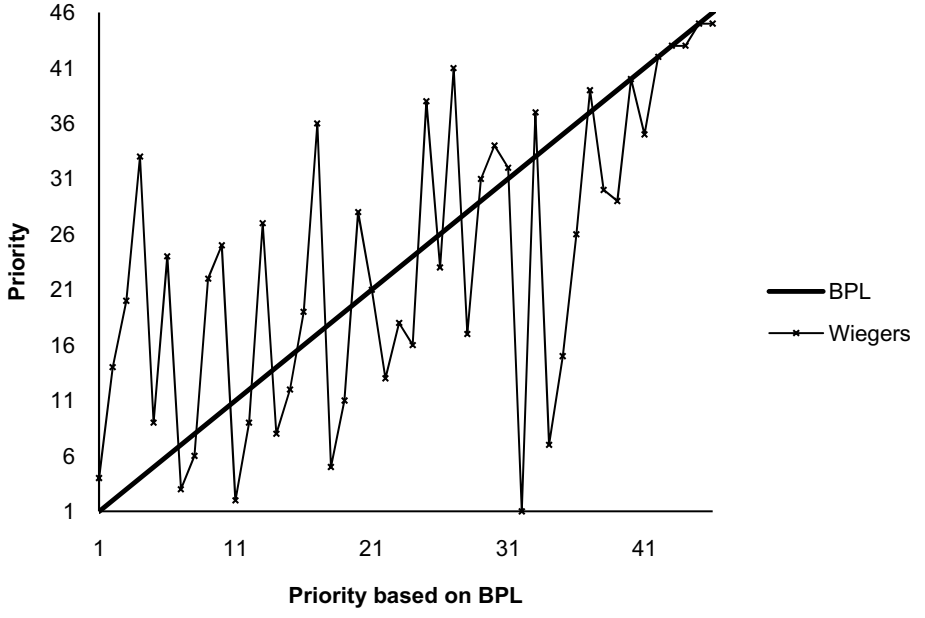


Fig. 4. CT Case: comparison of the two prioritization techniques

In general, in both case studies, the results from both techniques differed strongly from each other. Interestingly, however, in the second case study, the results of the two techniques are close to each other for the requirements with the lowest priority. The average difference of the priorities based on the two techniques was 14.75 in the first case study and 8.54 in the second case study. The maximal difference between BPL and Wiegers's technique was 56 in the first case and respectively 31 in the second. Both product managers rate Wiegers's technique rather low in terms of reliability (see Table 1). However, the product manager of Edmond noted that a better calibration might have resulted in an improvement. In their rating of ease of use of Wiegers's technique, the two product managers differ considerably. The product manager of CT mentioned that he found it difficult to estimate values for relative risk and penalty.

4.4 Discussion

The two techniques compared in these case studies differ in the way the prioritization criterion is articulated. In contrast to Wiegers's technique, BPL does not make the underlying inputs for the prioritization explicit. The results are rather based on the spontaneous intuition of the person applying it. However, ideally the product manager bases his considerations during the BPL prioritization on the same inputs, namely benefit, penalty, costs and risk as in Wiegers's technique or even considers other important factors, as for instance attractiveness for development.

In order to make sure that this happens, the question asked in the prioritization dialogue of the BPL tool should be formulated accordingly. During the case study the

question was “Is req. X more important than req. Y?” Now, we would suggest formulating it as “Do you want to implement req. X before req. Y?” instead. This formulation more explicitly suggests considering other factors than just importance. However, we still recommend avoiding prioritization of requirements that differ considerably in terms of costs to be implemented.

The difference in how explicitly the two techniques require the product manager to express the factors that determine his considerations also explains the different time consumption of the two techniques. The prioritization with BPL consequently only takes one quarter (Edmond case) to one third (CT case) of the time of Wiegers’s technique.

BPL was perceived the easier of the two techniques. This can also be related to its simple structure. The user basically only has to compare two requirements at the time and can apply an own set of criteria.

The strong difference between the two techniques’ prioritization suggests that the result of a prioritization session depends heavily on the respective technique used. The accordance of the two techniques’ results for requirements with low priority is actually the only point where both techniques correspond considerably with each other. It might be explained by the fact that the product manager considered the last four requirements as so unimportant that he assigned very low scores to three of the determining factors of Wiegers’s technique to them.

Both product managers considered the result of BPL considerably more reliable than that of Wiegers’s technique. This may be attributed to the fact that they could directly apply their own comparison criteria. As a consequence, the result always stays relatively close to his intuition. However, Wiegers’s technique might become more reliable when it is fine-tuned to the circumstances of the environment it is applied in by changing the weights of the four input factors. To test this, we would suggest repeating the prioritization with a small amount of requirements and subsequently adjust the weights in such a way that the prioritization result corresponds to the manual one.

Altogether, the results from the two case studies suggest that BPL is an appropriate technique to prioritize a few dozen requirements as they typically occur in small product software companies although we cannot generalize from the case studies due to the limited number of replication. In such an environment, the technique’s overall prioritization process quality, considering the quality of the process itself and the quality of its results, seems to be higher than that of Wieger’s technique. BPL could help small software product companies without a formal prioritization process to systemize it. The best results are expected when requirements are compared that are similar in terms of development costs. The technique was used by one single person. It remains open if it is also suitable with a group of people performing a prioritization as e.g. in the situation mentioned in the Edmond case study where three people are in charge of prioritizing the high-level requirements.

However, the case studies also revealed some limitations of the technique. First of all, BPL does not consider dependencies between requirements. Instead, the user has to keep them in mind while prioritizing or refining the prioritization list afterwards as also suggested in the requirements selection phase of the release planning as also indicated in the reference framework for software product management [3]. In addition, due to the simple structure there might be a tendency to base the prioritization

just on one single criterion, such as importance rather than consider other factors such as costs, penalty and risk.

In terms of scalability, the first case study revealed that a pair-wise comparison of 68 requirements can already be quite tiring and lead to mistakes in terms of the comparison. Balancing the binary tree and incorporating other BST optimization techniques [11] could reduce the number of comparisons necessary. However, we expect BPL not to be practicable for numbers much more than 100 requirements.

5 Conclusion and Further Research

The research question, as proposed in section 1, states:

How can BPL be applied as a requirement prioritization technique in small product software companies and how reliable are its results?

We conducted case studies in two small Dutch product software companies to validate the applicability of the technique. The product managers of these companies used the technique to prioritize a few dozen low-level requirements. To assess the reliability of the results they did the same with another well-known prioritization technique, the Wiegers matrix. Subsequently, the results of the two techniques were compared in terms of time consumption, ease of use and subjective reliability of the results. In both cases, BPL scored higher than Wiegers's technique in all aspects of the comparison.

We can conclude that BPL is a suitable technique for prioritizing medium amounts of requirements and could especially help smaller software product companies to formalize their requirements prioritization process. This finding complements research performed by Ahl [8] who compared BPL with other techniques in an experiment with a relatively small amount of requirements. BPL seems especially applicable when applied for prioritizing low-level requirements of similar size.

Some limitations of the technique that became apparent are (1) the missing consideration of dependencies between requirements and (2) the fact that the BPL prioritization might have the tendency to only be based on one criterion, as for instance benefit, rather than considering other factors (e.g. costs, penalty, and risk) as well. This effect might, however, be mitigated by improving the tool user-interface and making the user aware that he should base his comparison of two requirements on a number of criteria.

Besides the limitation that two case studies are not a large base of generalizability, they only investigated the applicability of the technique in prioritization performed by one person. Further research should examine if BPL can be used in a group prioritization, e.g. in situations that require a discussion between different stakeholders. It is likely that techniques making the prioritization inputs more explicit, such as the Wiegers matrix, are more appropriate for this kind of prioritization since they provide a ground for discussion.

We have already referred to the limitation concerning the comparability of requirements with considerably different costs. Further research could investigate what possibilities exist to further mitigate this limitation. One possibility would be to use BPL to explicitly prioritize requirements in terms of a number of factors, such as benefit, penalty, costs and risk, and weight these sub-results in order to compute an overall prioritization.

References

1. Xu, L., Brinkkemper, S.: Concepts of product software. European Journal of Information Systems 16(5), 531–541 (2007)
2. Berander, P., Khan, K.A., Lehtola, L.: Towards a Research Framework on Requirements Prioritization. In: Proceedings of the Sixth Conference on Software Engineering Research and Practise in Sweden, pp. 39–48 (2006)
3. van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: Towards a reference framework for software product management. In: Proceedings of the 14th International Requirements Engineering Conference, pp. 312–315 (2006)
4. Augustine, S.: Managing Agile Projects. Prentice Hall, New Jersey (2005)
5. Racheva, Z., Daneva, M., Buglione, L.: Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products. In: Proceedings of the Second International Workshop on Software Product Management 2008, Barcelona, pp. 49–58 (2008)
6. Karlsson, J., Wohlin, C., Regnell, B.: An evaluation of methods for prioritizing software requirements. Information and Software Technology 39(14-15), 939–947 (1997)
7. Wiegers, K.: First things first: prioritizing requirements. Software Developmen 7(9), 48–53 (1999)
8. Ahl, V.: An Experimental Comparison of Five Prioritization Methods. Master's Thesis. Department of Systems and Software Engineering, Blekinge Institute of Technology, Ronneby (2005)
9. Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Software 14(5), 67–74 (1997)
10. Knuth, D.: The Art of Computer Programming, vol. 3. Addison-Wesley, Reading (1997)
11. Knuth, D.: Optimum binary search trees. Acta Informatica 1(1), 14–25 (1971)
12. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
13. Bell, J., Gupta, G.: An evaluation of self-adjusting binary search tree techniques. Software: Practice and Experience 23(4), 369–382 (1993)
14. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. Management Information Systems Quarterly 28(1), 75–106 (2004)
15. Yin, R.K.: Case study research: Design and methods. Sage, Thousand Oaks (2009)
16. Wohlin, C., Wesslen, A.: Experimentation in software engineering: an introduction. Kluwer, Norwell (2000)
17. Smith, J.D.: Design and Analysis of Algorithms. PWS-KENT, Boston (1989)
18. Young, R.R.: Recommended requirements gathering practices, CrossTalk, pp. 9–12 (April 2002)
19. Herrmann, A., Daneva, M.: Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, pp. 125–134 (2008)
20. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall, Upper Saddle River (2001)
21. JIRA Bug tracking, issue tracking and project management software, http://www.atlassian.com/software/jira/
22. McConnell, S.: Rapid Development: Taming Wild Software Schedules, 1st edn. Microsoft Press, Redmond (1996)

Towards a Framework for Specifying Software Robustness Requirements Based on Patterns

Ali Shahrokni and Robert Feldt

Department of Computer Science & Engineering
Chalmers University of Technology
{nimli,robert.feldt}@chalmers.se

Abstract. [**Context and motivation**] With increasing use of software, quality attributes grow in relative importance. Robustness is a software quality attribute that has not received enough attention in requirements engineering even though it is essential, in particular for embedded and real-time systems. [**Question/Problem**] A lack of structured methods on how to specify robustness requirements generally has resulted in incomplete specification and verification of this attribute and thus potentially a lower quality. Currently, the quality of robustness specification is mainly dependent on stakeholder experience and varies wildly between companies and projects. [**Principal idea/results**] Methods targeting other non-functional properties such as safety and performance suggest that certain patterns occur in specification of requirements, regardless of project and company context. Our initial analysis with industrial partners suggests robustness requirements from different projects and contexts, if specified at all, follow the same rule. [**Contribution**] By identifying and gathering these commonalities into patterns we present a framework, *ROAST*, for specification of robustness requirements. ROAST gives clear guidelines on how to elicit and benchmark robustness requirements for software on different levels of abstraction.

1 Introduction

With software becoming more commonplace in society and with continued investments on finding better ways to produce it the maturity of both customers' requirements and our ability to fulfill them increases. Often, this leads to an increased focus on non-functional requirements and quality characteristics, such as performance, design and usability. But there is also less of a tolerance for faults and failures; by coming more reliant on software our society also increasingly requires it to be reliable and robust. *Robustness* as a software quality attribute (QA) is defined as [1]: "The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions."

The industrial project which is referred to in this paper is the development of a robust embedded software platform for running internal and third party telematics services [2]. The platform and services in this context need to be

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 79–84, 2010.

dependable in presence of erroneous input. Avoiding disturbance to a service or the platform by other services is another essential property. Considering the industrial context, robustness is interpreted as stability in presence of erroneous input and execution stability in presence of stressful environment created by external services or modules.

Robustness requirement specification is relatively unexplored in the academic literature. Much has happened in the software engineering field since the few papers that focus on robustness specification have been published [3]. However, There are common grounds between robustness and other more explored QA that can help understanding robustness better. Statements, requirements and checklists found in the literature about safety, security and dependability requirements can in some cases be applied to robustness too. This fact has been used to acquire a framework for robustness requirement (RR) specification in this paper. This commonality depends on the fact that lack of robustness is in most cases experienced and manifested as lack of other QAs or even functionality in the system. According to Lutz and Newmann [4,5], by including requirements for robustness or "defensive design" in the specifications many safety-related errors can be avoided. In this study Lutz shows that the majority of safety-related software errors in Voyager and Galileo spacecraft were interface(robustness) and functional errors.

Many failures associated with requirements are due to incompleteness [3]. This fact has motivated this study and act as a guideline to create a relevant framework that helps improving the quality of requirement specification process and ensures better completeness of the set of requirements.

In this paper we propose a framework called ROAST for specifying RR. This framework is the result of gathering data from different industrial and academic sources and applying it to the industrial project mentioned earlier. During the process a number of gaps in the existing work in the field of RR specification were identified. ROAST is the result of these steps and it can be used as a guide for RR specification and testing.

2 The ROAST Framework

In this section the framework ROAST for eliciting RR and aligning specification and testing of RR is shortly described. ROAST is based on identifying patterns for specification of robustness at different abstraction levels. As mentioned earlier, robustness is not a strictly defined term and can refer to both low-level (interface and input validation, failure handling) and high-level (service degradation, availability, reliability and dependability) requirement types.

There are three main ideas behind the method: (a) specification levels, (b) requirement patterns, and (c) alignment from requirements to testing. The first 2 parts are shortly described in this paper and the alignment from requirements to testing will be discussed in future publications.

Like many NFRs, RR are often summative in nature. This means that they specify general attributes of the developed system and not specific attributes for

specific, 'local' situations. For example, while a functional requirement (FR) for a certain feature of a telematics system ('system should support being updated with applications during runtime') can be judged by considering if that specific feature is present or not, a RR ('system should be stable at all times, it cannot shut down because of erroneous inputs or components') requires testing of a large number of different system executions. So while a FR talks about one specific situation, or a definite sub-set of situations, a RR summarizes aspects of the expected system behavior for a multitude of situations.

To make RRs testable they need to be refined into specific behaviors that should (positive) or should never happen (negative). Early in the development of a software system users or developers may not be able to provide all details needed to pinpoint such a behavior (or non-behavior). However, it would be a mistake not to capture more general RRs. Our method thus describes different information items in a full specification of a robustness behavior and describes different levels in detailing them. This is similar to the Performance Refinement and Evolution Model (PREM) as described by [6,7], but specific to robustness instead of performance requirements. The different levels can be used to judge the maturity of specifying a requirement or as a specific goal to strive for.

Since RR are often summative, i.e. valid for multiple different system situations, they are also more likely, than specific functional requirements, to be similar for different systems. We can exploit this similarity to make our method both more effective (help achieve a higher quality) and efficient (help lower costs). By creating a library of common specification patterns for robustness, industrial practitioners can start their work from that library. Thus they need not develop the requirements from scratch and can use the pattern to guide the writing of a specific requirement. This can both increase quality and decrease time in developing the requirements. Our approach and the pattern template we use is based on the requirements patterns for embedded systems developed by Konrad and Cheng, that are in turn based on the design patterns book [8,9].

The verification of different robustness behaviors should be aligned with the RR. Based on the level of requirement and the pattern the requirement is based on different verification methods are applicable and relevant. We make these links explicit in order to simplify the verification and testing process. Note that the verification method that is relevant for a certain pattern at a certain level may not actually be a testing pattern. For pattern levels that are not quantifiable or testable, the verification method may be a checklist or similar technique.

Figure 1 gives an overview of the method we propose, and shows both the levels, robustness areas with patterns and attached verification methods. In the following we describe each part in more detail.

2.1 Robustness Requirements Levels

To have a similar model for RR as presented in [7], we need to identify the factors that can affect robustness. These factors are generally not the same as the ones affecting performance. We also need to specify these factors in more detail for them to be useful for practitioners. Since these factors might be simulated at

many different levels of fidelity it is important to realize that level 2 is rather a continuum of lower or higher fidelity approximations of level 3. We also use our patterns to clarify different ways for quantifying qualitative requirements.One important such quantification is transforming a qualitative requirement for the system as a whole into a local variant specific to certain components and/or sub-systems. We have introduced named levels for these two different types of qualitative requirement types.

Figure 1 shows these different levels of a requirement in a diagram. The two main axis of this model are 'Specificity' and 'Realism'. The former increases when we detail the scope of the requirement, from a global, system-wide requirement to be localized to a component or sub-system. It also increases when we quantify how and to what degree the requirement is to be fulfilled. The 'Realism' axis increases when we mimic the realism of the factors that affects system execution and its robustness. When we do no or little specification of these factors the requirement is a RR-3 requirement, i.e. specific but not realistically specified. As we describe the factors more realistically we increase realism into a RR-4. We can strive to reach RR-5 by using real-world values and workloads in describing the factors.

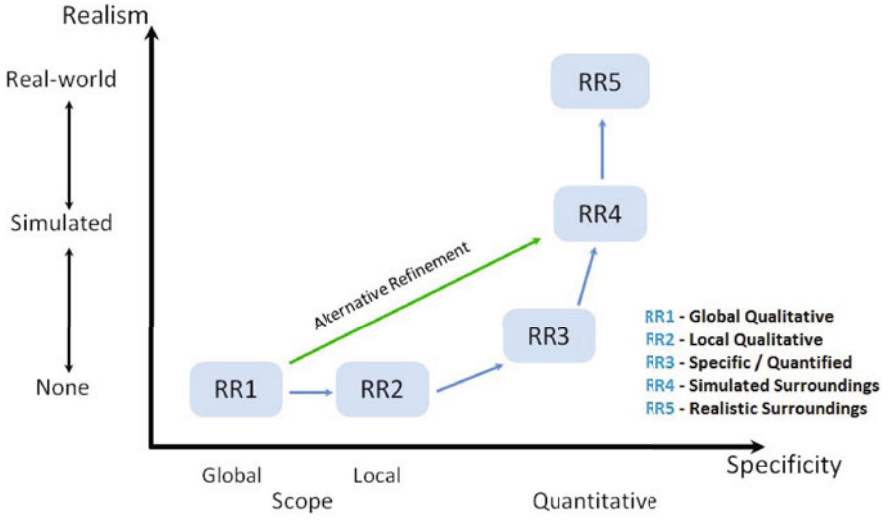


Fig. 1. Robustness Requirements Levels and Typical Refinement

2.2 Robustness Specification Patterns

Konrad and Cheng introduced requirements patterns for embedded systems [8]. They modify the original template introduced by Gamma et al by adding information items for 'Constraints', 'Behavior' and 'Design Patterns' and by deleting the 'Implementation' and 'Sample Code' items. The focus for Konrad and Cheng

is to specify the connection from requirements to the design as specified in UML diagrams. Even though this can be a worthy goal also for our method in the long-term we would rather keep design alternatives out of the robustness patterns. Primarily because multiple different designs will be able to support RR; pre-specifying the allowed solutions would be too restrictive for developers. The connection from the patterns to verification and test methods that we propose is more natural since each requirement will typically need to be tested and each verification activity should be motivated by some requirement. We have thus modified the template by Konrad and Cheng to reflect this difference in purpose. A pattern captures a whole family of related requirements but that can vary according to our levels.

Some of the patterns in ROAST are similar to the ones introduced by Lutz [4] which account for the majority of safety-related errors Galileo and Voyager However, when working on a platform in the presence of many services, with little runtime control, it is essential to predict not only how the system can be affected through interfaces but even how it behaves when sharing resources with other services. The 14 identified robustness patterns are presented in table 1 where *IS* is Input Stability, *ES* Execution Stability and *M* stands for Means to achieve robustness:

Table 1. Robustness Specification Patterns

N	Pattern	Category
1	Specified response to out-of-range and invalid inputs	IS
2	Specified response to timeout and latency	IS
3	Specified Response to input with unexpected timing	IS
4	High input frequency	IS
5	Lost events	IS
6	High output frequency	IS
7	Input before or during startup, after or during shut down	IS
8	Error recovery delays	IS
9	Graceful degradation	M
10	All modes and modules reachable	M
11	run-time memory access in presence of other modules and services	ES
12	Processor access in presence of other modules and services	ES
13	Persistent memory access in presence of other modules and services	ES
14	Network access in presence of other modules and services	ES

The patterns presented in this section are partly elicited by studying earlier requirement documents from similar projects and partly through expertise provided by the participants in the project who are mainly experienced people in the field of requirement engineering. Earlier academic work presented above helped us complete and reformulate already identified patterns.

3 Conclusion

The state of the art and practice concerning robustness requirements and testing is rather immature compared to that of other quality attributes. The proposed framework, ROAST, is a unified framework for how to interpret robustness and specify and verify robustness requirements.

ROAST follows a requirement as it often evolves from a high level requirement to a set of verifiable and concrete requirements. Therefore ROAST consists of different levels of specification that follow the most typical requirement specification phases practiced in the industry. As presented in ROAST, requirements engineering process tends to start from high level requirements and break them down into more specific and measurable ones. Therefore, ROAST can be incorporated into the activities of most companies with minimal change to the rest of the process. The commonality often seen between robustness requirements in different projects is captured in patterns. For different patterns and levels different verification methods will be more or less useful.

Initial evaluation of ROAST has been carried out in an industrial setting. Preliminary results are promising and show that the resulting requirements are more complete and more likely to be verifiable. Further evaluation is underway.

References

1. IEEE Computer Society, IEEE standard glossary of software engineering terminology. IEEE, Tech. Rep. Std. 610.12-1990 (1990)
2. Shahrokni, A., Feldt, R., Petterson, F., Back, A.: Robustness verification challenges in automotive telematics software. In: SEKE, pp. 460–465 (2009)
3. Jaffe, M., Leveson, N.: Completeness, robustness, and safety in real-time software requirements specification. In: Proceedings of the 11th International Conference on Software Engineering, pp. 302–311. ACM, New York (1989)
4. Lutz, R.R.: Targeting safety-related errors during software requirements analysis. Journal of Systems and Software 34(3), 223–230 (1996)
5. Newmann, P.: The computer-related risk of the year: weak links and correlated events. In: Proceedings of the Sixth Annual Conference on Computer Assurance, COMPASS 1991, Systems Integrity, Software Safety and Process Security, pp. 5–8 (1991)
6. Ho, C.-W., Johnson, M., Maximilien, L.W.E.: On agile performance requirements specification and testing. In: Agile Conference 2006, pp. 46–52. IEEE, Los Alamitos (2006)
7. Ho, C.-W.: Performance requirements improvement with an evolutionary model. PhD in Software Engineering, North Carolina State University (2008)
8. Konrad, S., Cheng, B.H.C.: Requirements patterns for embedded systems. In: Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE 2002), Essen, Germany (September 2002)
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley, Boston (January 1995)

A Metamodel for Software Requirement Patterns*

Xavier Franch[1], Cristina Palomares[1], Carme Quer[1], Samuel Renault[2], and François De Lazzer[2]

[1] Universitat Politècnica de Catalunya (UPC)
UPC – Campus Nord, Omega building, 08034 Barcelona (Spain)
{franch,cpalomares,cquer}@essi.upc.edu
[2] CITI, CRP Henri Tudor
29 avenue John F Kennedy, Luxembourg (Luxembourg)
{samuel.renault,francois.delazzer}@tudor.lu

Abstract. [Context and motivation] Software Requirement Patterns (SRP) are a type of artifact that may be used during requirements elicitation that also impact positively in other activities like documentation and validation. In our experiences, SRP show a great percentage of reuse for the non-functional requirements needed in call-for-tender requirement specifications. **[Question / problem]** We are facing the need of formulating the accurate definition of SRP for their use in call-for-tender processes to allow reasoning rigorously and know more about their semantics and applicability. **[Principal ideas / results]** In this paper we present a metamodel for SRP around three main concepts: 1) the structure of SRP themselves; 2) the relationships among them; 3) the classification criteria for grouping them. **[Contribution]** We provide a rigorous definition that shows the concepts that are of interest when defining and applying SRP.

Keywords: software requirement patterns, requirements reuse, metamodel.

1 Introduction

Reuse is a fundamental activity in all software development related processes. Of course, requirements engineering is not an exception to this rule [1]. The reuse of software requirements may help requirement engineers to elicit, validate and document software requirements and as a consequence, obtain software requirement specifications of better quality both in contents and syntax [2].

There are many approaches to reuse. Among them, patterns hold a prominent position. According to their most classical definition, each pattern describes a problem which occurs over and over again, and then describes the core of the solution to that problem, in such a way that it can be used a million times over, without ever doing it the same way twice [3]. Software engineers have adopted the notion of pattern in several contexts, remarkably related with software design (e.g., software design and architectural patterns), but also in other development phases, both earlier and later.

* This work has been partially supported by the Spanish project TIN2007-64753.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 85–90, 2010.

We are interested in the use of patterns for the software analysis stage, namely Software Requirement Patterns (SRP).

As [4] shows, there are not much proposals for SRP in the literature, in fact their exhaustive review lists just 4 catalogues out of 131, compared to 47 design catalogues and 39 architecture catalogues. Our own literature review has found some more approaches but still this unbalance is kept. The existing approaches differ in criteria like the scope of the approach, the formalism used to write the patterns, the intended main use of patterns and the existence of an explicit metamodel. Table 1 shows the classification of these approaches with respect to the mentioned criteria. In the last row we describe our own method as general-purpose, representing patterns in natural language, aiming at writing sofware requirements specifications (SRS) and metamodel-based.

About the two approaches that propose a metamodel, [8] focus on reuse of semi-formal models (e.g., UML class diagrams and sequence diagrams), thus the kind of concepts managed are quite different. Concerning [6], their focus is on variability modeling for handling the different relationships that requirements may have. From this point of view, it is a very powerful approach, but other aspects that we will tackle here, like the existence of different forms that a pattern may take, or multiple classification criteria, are not present in their metamodel.

Table 1. Comparison of approaches to software requirement patterns

	Scope	*Notation*	*Application*	*Metamodel?*
[5]	General purpose	Natural language	Req. elicitation	Just templates
[6]	General purpose	Object models	Variability modeling	Yes
[7]	Business applications	Event-Use case	Identify patterns	No
[8]	General purpose	Semi-formal models	Writing req. models	Yes
[9]	Embedded systems	Logic-based	From informal to formal reqs.	No
[10]	Security requirements	UML class diagrams	Security goals elicitation	No
[11]	Security requirements	Natural language	Req. elicitation in SOC	No
[12]	General purpose	Natural language	Writing SRS	Just template
[13]	General purpose	Problem frames + *i**	Knowledge management	No
Ours	General purpose	Natural language	Writing SRSs	Yes

The idea of using SRP for reusing knowledge acquired during this stage arose from the work of the CITI department of the Centre de Recherche Publique Henri Tudor (CRPHT) on helping SME with no background in requirements engineering to handle requirements analysis activities and to design SRS in order to conduct call-for-tender processes for selecting Off-The-Shelf (OTS) solutions [14]. More than 40 projects ran successfully following the CITI methodology, but the only technique of reuse they applied was starting a new project by editing the most similar requirement book. These techniques demonstrated their weaknesses especially in relation to mobility of IT experts and consultants. It became necessary to provide better means to capitalize requirements in a high-level manner by creating reusable artifacts like patterns, supporting consultants' need of creating new SRS.

As a response to this need, we built an SRP catalogue with 29 patterns. The patterns were all about non-functional requirements since this type of requirements are the less sensitive to changes in the problem domain. The research method used to build this catalogue and the underlying metamodel was based on the study of SRS from 7

call-for-tender real projects conducted by CITI; experts' knowledge, being these experts: IT consultants, CITI facilitators and UPC researchers; background on requirements engineering literature and especially on requirement patterns. We undertook then a first validation in two real projects. In this paper we focus on the metamodel, that is, the structure of our proposed SRPs and its classification to facilitate the selection of patterns. The PABRE process of application of SRP in the context of CITI and the validation of our current SRP catalogue have been described in [15], therefore neither the process nor the catalogue's content are part of the objectives of this paper.

2 Structure of a Requirement Pattern

The first fundamental question to answer is what the structure of a SRP is. Figure 1 shows an example of SRP that illustrates the most significant components. Note the statement of the goal as a kind of problem-statement of the pattern; goals play a crucial part in the PABRE method built on top of these patterns [15]. SRP metadata (e.g., description, author) are not included for the sake of brevity.

Requirement Pattern Failure Alerts			
Goal Satisfy the customer need of having a system that provides alerts when system failures occur			
Requirement Form *Heterogeneous Failure Alerts*	**Fixed Part**	**Template**	*The system shall trigger different types of alerts depending on the type of failure*
		Extended Parts Constraint	*multiplicity(Alerts for Failure Types) = 0..**
	Extended Part *Alerts for Failure Types*	**Template**	*The system shall trigger %alerts% alerts in case of %failures% failures*
		Parameter	**Metric**
		alerts: non-empty set of alert types	*alerts: Set(AlertType)* *AlertType: Domain of possible types of alerts*
		failures: non-empty set of failure types	*failures: Set(FailureType)* *FailureType: Domain of possible types of failures*
Requirement Form *Homogeneous Failure Alerts*	**Fixed Part**	**Template**	*The system shall trigger an alert in case of failure.*
		Extended Parts Constraint	*multiplicity(AlertsTypes) = 0..1 and* *multiplicity(Failure Types) = 0..1*
	Extended Part *Alert Types*	**Template**	*The solution shall trigger %alerts% alerts in case of failure*
		Parameter	**Metric**
		alerts: non-empty set of alert types	*alerts: Set(AlertType)* *AlertType: Domain of possible types of alerts*
	Extended Part *Failure Types*	**Template**	*The system shall trigger alerts in case of %failures% failures*
		Parameter	**Metric**
		failures: non-empty set of failure types	*failures: Set(FailureType)* *FailureType: Domain of possible types of failures*

Fig. 1. An example of software requirement pattern (parameters appear among '%')

Figure 2 shows the metamodel for SRP. It represents the metaclasses for the basic concepts that appear in the example above and others introduced later. We may observe that the concept represented by a *Requirement Pattern* may take different *Pattern Forms*. Each form is applicable in a particular context, i.e. it is the most appropriate form to achieve the pattern's goal in a particular type of software project. In the example of Fig. 1, the second form is more adequate if the types of alerts that the client wants in the system will be the same for all types of failures, if not the first form must be applied. Applying a SRP, then, means choosing and applying the most suitable form.

At its turn, each form has a *Fixed Part* that characterizes it which is always applied if the form is selected, together with zero or more *Extended Parts* that are optional and help customizing the SRP in the particular project. In general, extended parts must conform to some *Constraint* represented by means of a formula over some predefined operators (e.g., for declaring multiplicities or dependencies among extended parts, as *excludes*, *requires*). For instance, in the example we may see that the first form allows repeated application of its single extended part, whilst the second form allows one application at most of each of its extended parts (since in this form it has not sense to state more than once the types of alerts and failures).

Both fixed and extended parts are atomic *Pattern Items* that cannot be further decomposed. Each pattern item contains a *template* with the text that finally appears in the SRS when applied. In this text, some variable information in the form of *Parameters* may (and usually, do) appear. Parameters establish their *Metric*, eventually a correctness condition *inv*, and also may be *related* to other parameters (belonging to other patterns) such that they must have the same value; an example is the parameter *failures* that also appears in some form of other SRP in the catalogue, namely the pattern *Recovery Procedures*.

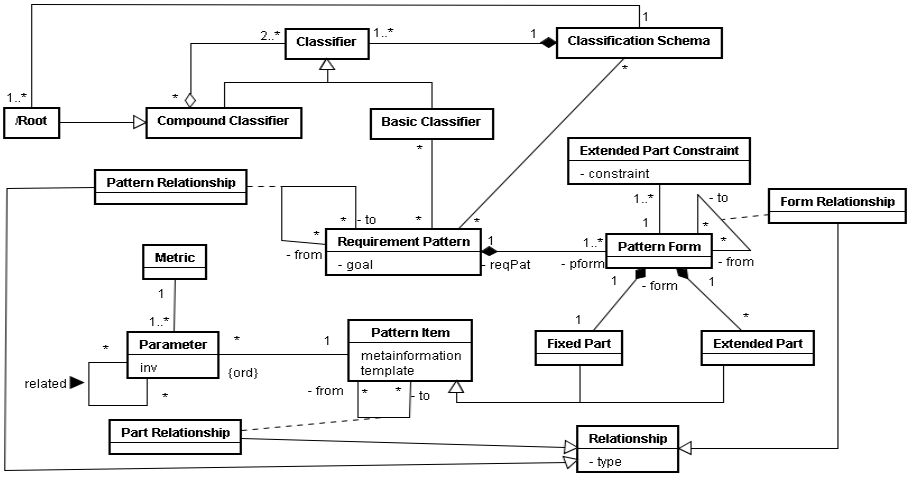


Fig. 2. The metamodel for software requirement patterns

SRPs are not isolated units of knowledge, instead there are several types of relationships among them. For instance, Withall structures his SRP catalogue using a more detailed proposal of relationships, that may be purely structural like "has", "uses" and "is-a", or with a semantic meaning like "displays" and "is across" [12]. Even generic (unlabelled) relationships are used. A thorough analysis of the SRS written by CITI shows that relationships may appear at three different levels:

- *Pattern Relationship*. The most general relationship that implies all the forms and all the forms' parts of the related patterns.
- *Form Relationship*. A relationship at the level of forms implies all the parts of the related forms.
- *Part Relationship*. The relationship only applies to these two parts.

In any case, if *A* is related to *B* and *A* is applied in the current project, the need of applying or avoiding *B* must be explicitly addressed. The types of relationships are not predetermined in the metamodel to make it more flexible. The superclass *Relationship* includes an attribute to classify each relationship.

A fundamental issue when considering patterns as part of a catalogue is the need of classifying them over some criteria for supporting their search. In fact, it is important to observe that different contexts (organizations, projects, standards, etc.) may, and usually do, define or require different classification schemas. History shows that trying to impose a particular classification schema does not work, therefore we decouple SRPs and *Classifiers* as shown in the metamodel. The catalogue is thus considered as flat and the *Classification Schemas* just impose different structuring schemas on top of it. Classifiers are organized into a hierarchy and then SRP are in fact bound to *Basic Classifiers*, whilst *Compound Classifiers* just impose this hierarchical structure. The use of aggregation avoids cycles without further integrity constraints. Last, a derived class *Root* is introduced as a facilitation mechanism.

The metamodel shows that a SRP may be bound to several classification schemas, and even to more than one classifier in a single classification schema (since no further restrictions are declared). Also note that we do not impose unnecessary constraints that could lead the catalogue to be rigid. For instance, we may mention that a classification schema may not cover all existing SRP (i.e., some SRP may not be classified). Although this situation could be thought as a kind of incompleteness, in fact we are allowing having dedicated classification schemas for particular categories of patterns, e.g. a performance classification schema, a classification schema just for the non-technical criteria [16] and then allowing to compound them for having a multi-source global classification schema. Also we remark that the PABRE method [15] benefits from this existence of multiple classification schemas since nothing prevents changing from one schema to another during catalogue browsing.

3 Conclusions and Future Work

In this paper we have presented a metamodel for software requirement patterns (SRP). This metamodel is the natural evolution of the preliminary proposal of SRP presented at [17] and shows the current concepts used by the PABRE method [15]. The metamodel helps to fix the concepts behind our proposal of SRP, improving the quality of the current SRP catalogue and process and has been be taken as starting point of the data model of an ongoing support tool. The metamodel has been validated with

respect to several software requirement specifications (SRS) written by CITI-CRPHT in the context of call-for-tender processes as well as in two processes themselves. The contents of the catalogue have been validated as explained in [15]; the catalogue itself can be found at the website http://www.upc.edu/gessi/PABRE.

Future work spreads over three main directions. Concerning validation, we are planning to run new case studies to debug all the PABRE components: metamodel, catalogue contents and process. We intend to experiment deeper the application of the SRP catalogue in several contexts (public IT procurement projects and Small- and Medium-Sized companies' projects). We also want to study the suitability of the current presented metamodels for other types of requirement patterns, that is, patterns for functional and non-technical requirements. Last, we will analyze the possibility of converting the current metamodel of a SRP catalogue into a metamodel for a patterns language which would eventually make possible the adoption of the approach in contexts with different needs than those presented here. Although the idea is appealing, it would require more engineering effort and thus needs careful analysis.

References

1. Lam, W., McDermid, J.A., Vickers, A.J.: Ten Steps Towards Systematic Requirements Reuse. REJ 2(2) (1997)
2. Roberson, S., Robertson, J.: Mastering the Requirements Process, 2nd edn. Addison-Wesley, Reading (2006)
3. Alexander, C.: The Timeless Way of Building. Oxford Books (1979)
4. Henninger, S., Corrêa, V.: Software Pattern Communities: Current Practices and Challenges. In: PLoP 2007 (2007)
5. Durán, A., Bernárdez, B., Ruíz, A., Toro, M.: A Requirements Elicitation Approach Based in Templates and Patterns. In: WER 1999 (1999)
6. Moros, B., Vicente, C., Toval, A.: Metamodeling Variability to Enable Requirements Reuse. In: EMMSAD 2008 (2008)
7. Robertson, S.: Requirements Patterns Via Events/Use Cases. In: PLoP 1996 (1996)
8. López, O., Laguna, M.A., García, F.J.: Metamodeling for Requirements Reuse. In: WER 2002 (2002)
9. Konrad, S., Cheng, B.H.C.: Requirements Patterns for Embedded Systems. In: RE 2002 (2002)
10. Matheson, D., Ray, I., Ray, I., Houmb, S.H.: Building Security Requirement Patterns for Increased Effectiveness Early in the Development Process. In: SREIS 2005 (2005)
11. Mahfouz, A., Barroca, L., Laney, R.C., Nuseibeh, B.: Patterns for Service-Oriented Information Exchange Requirements. In: PLoP 2006 (2006)
12. Withall, J.: Software Requirements Patterns. Microsoft Press, Redmond (2007)
13. Yang, J., Liu, L.: Modelling Requirements Patterns with a Goal and PF Integrated Analysis Approach. In: COMPSAC 2008 (2008)
14. Krystkowiak, M., Bucciarelli, B.: COTS Selection for SMEs: a Report on a Case Study and on a Supporting Tool. In: RECOTS 2003 (2003)
15. Renault, S., Méndez, O., Franch, X., Quer, C.: A Pattern-based Method for building Requirements Documents in Call-for-tender Processes. IJCSA 6(5) (2009)
16. Carvallo, J.P., Franch, X., Quer, C.: Managing Non-Technical Requirements in COTS Components Selection. In: RE 2006 (2006)
17. Méndez, O., Franch, X., Quer, C.: Requirements Patterns for COTS Systems. In: ICCBSS 2008 (2008)

Validation of the Effectiveness of an Optimized EPMcreate as an Aid for Creative Requirements Elicitation

Victoria Sakhnini[1], Daniel M. Berry[1], and Luisa Mich[2]

[1] Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, N2L 3G1 Canada
vsakhnin@uwaterloo.ca, dberry@uwaterloo.ca
[2] Department of Computer and Management Sciences, University of Trento
I-38100 Trento, Italy
luisa.mich@unitn.it

Abstract. [Context and Motivation] Creativity is often needed in requirements elicitation, and techniques to enhance creativity are believed to be useful. **[Question/Problem]** This paper describes a controlled experiment to compare the requirements-elicitation effectiveness of three creativity enhancement techniques: (1) full EPMcreate; (2) Power-Only EPMcreate, an optimization of full EPMcreate; and (3) traditional brainstorming. **[Principal ideas/Results]** Each technique was used by teams of students each of which applied its technique to generate ideas for requirements for enhancing a high school's public Web site. **[Contribution]** The results of this first experiment indicate that Power-Only EPMcreate is more effective, by the quantity and quality of the ideas generated, than full EPMcreate, which is, in turn, more effective than brainstorming.

1 Introduction

Many have observed the importance of creativity in requirements engineering, e.g., [1,2,3]. Many techniques, e.g., brainstorming [4], Six Thinking Hats [5], and the Creative Pause Technique [6], have been developed to help people be more creative. Some of these techniques have been applied to requirements engineering [7,2], and some of these techniques have also been subjected to experimental validation of their effectiveness [7,8]. A fuller discussion of these techniques can be found elsewhere [9].

This paper investigates a variant of the creativity enhancement technique (CET), *EPMcreate (EPM Creative Requirements Engineering [A] TEchnique)* [9,10], that is based on the *Elementary Pragmatic Model (EPM)* [11] and on a general-purpose CET developed to increase individual creativity [12]. The feasibility of applying EPMcreate to idea generation in requirements elicitation was established by experiments on two computer-based system (CBS) development projects with very different characteristics. Each experiment compared the requirements idea generation of two analysis teams, one using EPMcreate and the other using brainstorming [9]. Because EPMcreate was a new CET being applied to requirements elicitation, it had been necessary to define both the input of a requirements elicitation session with EPMcreate and the process. The main inputs of such a session are:

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 91–105, 2010.

- the problem statement or any other information useful for the CBS to be developed, and
- an understanding of the viewpoints of different stakeholders of the CBS, as defined by EPM, i.e., based on a systematic enumeration of all possible combinations of the stakeholders' viewpoints.

The definition of the process describes the steps and the activities to be performed at each step.

Effectiveness was chosen as the first research question: Is EPMcreate at least as effective as brainstorming? Brainstorming was chosen as the basis for a comparative measure of effectiveness, because: (1) it is well known [1,13]; and (2) there are at least two studies of its application in requirements elicitation [7,14], one experimental and the other anecdotal.

The results of the first experiments confirmed that, in at least the situations of the experiments, EPMcreate:

1. can be used by analysts, both junior and senior, requiring only minimal training and
2. produces more ideas and, in particular, more innovative ideas than does brainstorming.

Another investigation [10] compared the quality of the ideas produced by the two treatments in these same experiments and concluded that EPMcreate produced more ideas related to content and service requirements than did brainstorming.

The first experiments exposed a number of issues to be explored in the future. These include:

> Are there optimizations of EPMcreate, which involve fewer steps than EPMcreate, that are at least as effective as EPMcreate in helping to generate ideas for requirements for CBSs?

Since an optimization of EPMcreate requires fewer steps than the full EPMcreate, if the optimization is only at least as effective as the full EPMcreate, the optimization is still an improvement.

The purpose of this paper is to take up this question. This paper describes one optimization of EPMcreate and demonstrates its effectiveness as a creativity enhancement technique (CET). It reports on a controlled experiment that compares the optimization with both the original EPMcreate and brainstorming when they are used to help elicit requirements for an improved version of a Web site.

In the rest of this paper, Section 2 describes the EPMcreate technique, including the optimization. Section 3 describes the experiment, including its hypotheses and its steps. Section 4 gives the results of the experiment, analyzes them and determines if the hypotheses are supported. Section 5 discusses limitations of the results, and Section 6 concludes the paper.

2 The EPMcreate Technique

A page limitation forces the description of EPMcreate in this section to be brief. However, EPMcreate is described fully elsewhere [9].

2.1 Basic, Full EPMcreate

EPMcreate supports idea generation by focusing the search for ideas on only one logical combination of two stakeholders' viewpoints at a time. Sixteen such combinations are possible, each corresponding to one of the Boolean functions, fi for $0 \leq i \leq 15$, of two variables. Some representative function names and their corresponding tables, in which "Vn" means "Stakeholder n's Viewpoint" and "C" means "combination", are:

$f0$

$V1$	$V2$	C
0	0	0
0	1	0
1	0	0
1	1	0

$f3$

$V1$	$V2$	C
0	0	0
0	1	0
1	0	1
1	1	1

$f5$

$V1$	$V2$	C
0	0	0
0	1	1
1	0	0
1	1	1

$f10$

$V1$	$V2$	C
0	0	1
0	1	0
1	0	1
1	1	0

$f15$

$V1$	$V2$	C
0	0	1
0	1	1
1	0	1
1	1	1

The interpretation of these tables in terms of combining the viewpoints of stakeholders SH1 and SH2 are:

- $f0$ represents disagreeing with everything, independently of either stakeholder's viewpoint.
- $f3$ represents agreeing with SH1 completely.
- $f5$ represents agreeing with SH2 completely.
- $f10$ represents disagreeing with SH2 completely, independently of SH1's viewpoint.
- $f15$ represents agreeing with everything, independently of either stakeholder's viewpoint.

If there are more than two stakeholders, the technique can be applied several times, for each relevant pair of stakeholders.

2.2 EPMcreate in Practice

EPMcreate can be applied in any situation in which ideas need to be generated, e.g., at any time that one might apply a CET, such as brainstorming. EPMcreate is by no means the only technique for identifying requirements; it is but one of many that can be used.

When a requirements elicitor (elicitor) determines that EPMcreate is an appropriate technique during requirements engineering for a CBS under consideration, she first chooses two kinds of stakeholders, SH1 and SH2, usually users of the CBS, as those whose viewpoints will be used to drive the application of EPMcreate. She may ask the CBS's analysts for assistance in this choice. She then convenes a group of these analysts. Figure 1 shows a diagram that the elicitor will show the chosen stakeholders as part of her explanation of EPMcreate. In this diagram, the two ellipses represent two different stakeholders' viewpoints. Thus, for example, the intersection region represents the stakeholders' shared viewpoints.

The elicitor tells all convened,

> *Today, we are going to generate requirement ideas in 16 idea generation steps. In each step, all of you will pretend to think from the viewpoints of two stakeholders,* SH1 *and* SH2*, and for each viewpoint,*

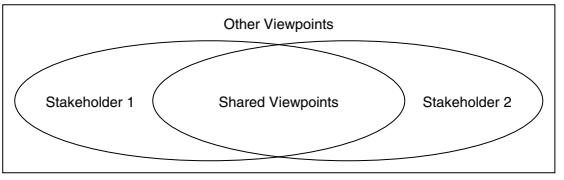


Fig. 1. Venn Diagram of Two Stakeholders' Viewpoints

- *In Step 0, you will blank out your minds.*
- *In Step 1, you will try to come up with ideas for problem solutions that are needed by both* SH1 *and* SH2*.*
- *In Step 2, you will try to come up with ideas for problem solutions that are needed by* SH1 *but not by* SH2*.*
- *In Step 3, you will try to come up with ideas for problem solutions that are needed by* SH1 *without concern as to whether they are needed by* SH2*.*
- *In Step 4, you will try to come up with ideas for problem solutions that are needed by* SH2 *but not by* SH1*.*

  ...
- *In Step 8, you will try to come up with ideas for problem solutions that are needed neither by* SH2 *nor by* SH1*, but are needed by other stakeholders.*

  ...
- *In Step 15, you will try to come up with ideas for problem solutions without concern as to whether they are needed by either* SH1 *or* SH2*.*

In the event that the elicitor believes that more than two stakeholders' viewpoints should be considered, she will convene more EPMcreate sessions, one for each pair of stakeholder viewpoints she believes to be useful. Her experience tells her how to identify subsets of stakeholders and stakeholder pairings that will yield the most new ideas for the fewest pairs.

2.3 Power Only EPMcreate

The optimization of EPMcreate that the research described in this paper studied is called the "Power-Only EPMcreate (POEPMcreate)", because it does only the four steps whose names are powers of two, namely $f1$, $f2$, $f4$, and $f8$. Their tables are:

$f1$

$V1$	$V2$	C
0	0	0
0	1	0
1	0	0
1	1	1

$f2$

$V1$	$V2$	C
0	0	0
0	1	0
1	0	1
1	1	0

$f4$

$V1$	$V2$	C
0	0	0
0	1	1
1	0	0
1	1	0

$f8$

$V1$	$V2$	C
0	0	1
0	1	0
1	0	0
1	1	0

This optimization, which does only four of the 16 original steps, was theorized to be at least as effective as the full EPMcreate, because the Boolean function of each of the rest of the steps can be achieved as a possibly empty disjunction of the functions of these four power-of-two steps.

3 Experiment Design and Planning

The main objective of this paper is to demonstrate the effectiveness of POEPMcreate as a creativity enhancement technique (CET). Since the effectiveness of EPMcreate was demonstrated by comparing it to brainstorming, the effectiveness of POEPMcreate is demonstrated by comparing it to each of EPMcreate and brainstorming. Therefore, we conducted an experiment and compared the requirement ideas for one CBS generated by six groups, two of which used POEPMcreate, two of which used EPMcreate, and two of which used brainstorming. The same number of subjects participated in the experiment for the same amount of time in each group. Each group was to generate, using its assigned CET, ideas for requirements for an improved version of an existing Web site [15]. The Web site was that of a Canadian high school with information directed to students, parents, teachers, and administrators. The site was chosen for its accessibility, lack of intellectual property restrictions, and the fact that as educators, the authors could be considered domain experts. We decided that the two types of stakeholders whose viewpoints would be adopted by the EPMcreate and POEPMcreate groups were students and parents. The quantity and quality of the requirement ideas generated by every group were evaluated.

3.1 Hypotheses

The hypotheses to be tested with the experiment were:

H1. The POEPMcreate is more effective than the full 16-step EPMcreate in helping to generate requirement ideas.

H2. The full 16-step EPMcreate is more effective than brainstorming in helping to generate requirement ideas.

Note that H1 is stronger than needed since all we require is that, as an optimization, the POEPMcreate is *at least* as effective as the full EPMcreate. Therefore, if H1 were not supported, it would be acceptable if its corresponding null hypothesis, that there is no difference in the effectiveness of the two CETs, were supported. As it turned out, H1 *is* supported. However, it is always nice when an optimization proves to be better than required.

We considered the hypothesis H2 because the original experiments [9] addressing this same hypothesis did not get generalizable results, although the results were significant for the CBSs and subjects studied in the experiments. The generalization is addressed by testing the effectiveness of EPMcreate to help generate requirement ideas for a different CBS with different subjects.

3.2 Measuring the Effectiveness of a CET

The effectiveness of a CET is measured by two numbers about the ideas generated when using the CET,

1. the quantity, i.e., the raw number, of ideas and
2. the number of high quality ideas.

The raw number of ideas generated was used because one of the CETs evaluated, brainstorming, encourages quantity over quality in its first step.

The basis for evaluating the quality of an idea is the notion that a creative idea is both new and useful [8]. Therefore, as suggested by Mich et al. [9], the quality of an idea was evaluated by classifying it into one of 4 rankings:

1. new and realizable
2. new and not realizable
3. not new and not realizable
4. not new and realizable

with 1 being the highest ranking and 4 being the lowest ranking.

An idea is considered new if the idea is not already implemented in the current Web site. "Realizable" includes several notions: (1) useful for at least one stakeholder, (2) technically implementable, and (3) socially and legally implementable, thus also excluding privacy invading ideas.

This ranking embodies three independent assumptions:

- that a new idea is better than a not new idea,
- that a realizable idea is better than a not realizable idea, and
- among the not new ideas, a not realizable one is more creative since it is more outside the box[1].

To evaluate the quality of the ideas, each of two domain experts, namely the first two authors of this paper, independently classified each idea into one of 4 rankings. In order to reduce the chances that the authors' desired results might affect the quality evaluation, we merged the requirement ideas generated by the 6 groups into one file. We then sorted the ideas alphabetically to produce the list of ideas to be evaluated. With the merged and sorted list, it was impossible for any evaluator to see which group, with its known CET, generated any idea being evaluated. After each evaluator had assigned a ranking to each idea, the rankings were copied to the original idea files, in order to be able to evaluate the quality of the requirement ideas of each group separately.

3.3 Steps of the Experiment

The steps for the experiment and their approximate times were:

Step 1: 20 minutes for each subject to filling a general information form, to allow us to know his or her background: The form included questions about his or her age, gender, native language, computer science (CS) courses, qualifications related to CS, employment history in CS, and knowledge of the CETs: brainstorming, EPMcreate, and POEPMcreate.

[1] Some might disagree with the ordering of Ranks 3 and 4. However, past experiments used this ordering of the ranks, and we needed to maintain consistency with the past experiments. In any case, only ideas receiving Ranks 1 and 2 were considered high quality. Therefore, the ordering of Ranks 3 and 4 has no effect on the results. Of course, in retrospect, "high quality" means simply "new". However, at the time we planned the experiment, we did not know exactly which ranks would be considered high quality.

Step 2: 30 minutes for each subject to take a creativity assessment test, the modified Williams test described in Section 3.4.

Step 3: 10 minutes for us to deliver to each group an explanation about the experiment and the CET it was to use:

- To the EPMcreate and POEPMcreate groups, the explanation was basically the last two paragraphs of Section 2.2 of this paper. The full-EPMcreate groups were given the full list of steps, and the POEPMcreate groups were given only Steps 1, 2, 4, and 8.
- To the brainstorming groups, the explanation emphasized that the main goal of a brainstorming session is to generate as many ideas as possible. Our recommendations and requests were:
  1. *Don't judge, be open to all ideas, and consider them carefully and respectfully in an unprejudiced manner.*
  2. *Encourage the unusual with no limits placed on your imaginations.*
  3. *The more ideas you generate, the better.*
  4. *Improve on the ideas of others; no one is the exclusive owner of any idea.*
  5. *Try to produce as many ideas as possible; don't evaluate any idea and don't inhibit anyone from participating.*

  Each group was told that it had only 120 minutes for its session and that it could finish session earlier if its members agreed that no other aspects could be discussed. Then, each group was given a short training session, with practice, about the CET it was to use.

Step 4: 120 minutes for each group to carry out its requirements elicitation session using the group's CET: Each group consisted of 4 subjects and was provided with two laptops: one to access the Web site that the group was to improve, and the other to write the requirement ideas generated by the group.

Each group, except one, used the full 120 minutes for requirements elicitation. Group 5, a brainstorming group, finished 25 minutes early, claiming that its members could not generate any more new ideas, but it was not the group that generated the fewest ideas.

Steps 1 and 2 were to be done in one 50-minute meeting and privately with each respondent to our advertisement for subjects, and Steps 3 and 4 were to be done in sessions attended by one or more groups.

3.4 Assigning Subjects into Balanced Groups

To find subjects, we asked hundreds of University of Waterloo undergraduate and graduate students by e-mail or in person to participate in the experiment for an honorarium of $20.00 (Canadian). Twenty-six students replied, and of these, 24 ended up being subjects in the experiment. We planned for 24 subjects, to make 6 groups of 4 each, plus 2 alternates in case a promised subject did not show. As it turned out, both alternates were needed.

Among the 24 subjects, 8 were female and 18 were male. The ages of the 24 ranged from 17 through 39. All the subjects had taken several Computer Science (CS) courses; 12 had taken more than 10 courses, and 12 had taken 3–5 courses. Sixteen subjects had

some professional work experience, and 8 did not. Eleven subjects had English as a native language, and 13 did not. All the subjects were familiar with brainstorming, but none had heard about any form of EPMcreate. Therefore, one might expect the groups using brainstorming to have an advantage; as it turned out, any such advantage proved to be of no help.

Six groups were created for the experiment: Groups 1 and 2 used POEPMcreate as their CET, Groups 3 and 4 used the full 16-step EPMcreate as their CET, and Groups 5 and 6 used brainstorming as their CET.

In order to create homogeneous groups in the experiment with equivalent spreads of CS knowledge, English fluency, work experience, and native creativity, we used data we had gathered about each subject in Steps 1 and 2. The data that we used were the number of CS courses the subject had taken, the subject's native language, whether the subject had worked professionally, and the results of the subject's taking an adult version of Frank Williams's Creativity Assessment Packet [16], hereinafter called the *Williams test*.

As in past experiments [9,17], the Williams test was administered to each subject to measure his or her native creativity. The subjects' test scores were to be used ensure that any observed differences in the numbers of ideas were *not* due to differences in the native creativity of the subjects. In order to avoid having to interpret specific scores, we used the subjects' Williams test scores as one of the factors to consider in forming knowledge-, skill-, experience-, and native-creativity-balanced groups.

The properties of the groups are shown in Table 1. Note that the average Williams test scores for the 6 groups were in the small range from 70.25 to 71.6 out of a possible 100. We did not consider gender or age in creating the groups because it would have been very difficult to balance these factors while balancing the other factors. Moreover, we did not believe that these factors are relevant; and even if they are, they are probably less relevant than the ones we did consider. Note that the table shown is not exactly the original table calculated during the formation of the groups; it is the final table produced taking into account the two alternates that replaced the two original assigned subjects that did not show. Fortunately, the alternates did not change the balance of the groups.

Table 1. Characteristics of Groups and Their Subjects

Group	Technique	# Males	# Females	# native in English	# not native in English	# taken ≥ 10 CS courses	# taken 3–5 CS courses	# worked professionally	# not worked professionally	Average age	Average age Williams test score
1	POEPMcreate	3	1	1	3	2	2	2	2	25.5	70.66
2	POEPMcreate	2	2	2	2	2	2	3	1	23.8	71.00
3	EPMcreate	2	2	2	2	1	3	3	1	21.5	70.75
4	EPMcreate	3	1	1	3	2	2	1	3	23.4	70.60
5	Brainstorming	4	0	3	1	1	3	1	3	20.2	71.60
6	Brainstorming	2	2	2	2	3	1	3	1	25	70.25

After assigning the subjects to each of the 6 groups, we drew lots for the groups to determine which groups were going to use each of the three CETs. The first two columns of Table 1 show the resulting assignment of groups to CETs.

Each group contained the same number of subjects and participated in its session for the same period of time so that the resources for the all groups would be the same. We tried to schedule all the groups into one session, but could not find a single time slot that all could attend. So we allowed each group to choose a time slot that was convenient for all of its members. So, while the session times for the groups differ, all times were at the subjects' convenience. Thus, we believe that differences in the subjects' moods and energy levels that might arise from differences in session times were minimized. Finally, all groups generated requirement ideas for the same Web site. Thus, only the numbers and quality of the creative ideas generated need to be compared in order to compare the effectiveness of the three CETs.

4 Experimental Results and Analysis

The experiment was conducted in 4 sessions: Group 2 and Group 3 met on 16 November 2009, Group 6 met on 17 November, Group 1 and Group 4 met on 18 November, and Group 5 met on 20 November. The sessions went well and offered no surprises. In particular, the Web site underwent no change in between the first and last sessions.

The rooms were big enough, and each group worked quietly enough that no group benefited from another. Moreover, since Steps 1 and 2 were done with each subject privately, no subject knew another subject until the subjects of a group met each other at the beginning of its elicitation session in which it did Steps 3 and 4. Thus, there was no information flow between subjects that could have affected the results.

The quantity and quality of the requirement ideas that were generated by the 6 groups, using EPMcreate, POEPMcreate, and brainstorming as CETs were evaluated, as described in Section 3.

4.1 Evaluation of the Quantity of the Requirement Ideas

As is shown in Figure 2,

- the two POEPMcreate groups generated 74 and 76 ideas,
- the two EPMcreate groups generated 63 and 60 ideas, and
- the two brainstorming groups generated 47 and 36 ideas.

These data show differences between the CETs.

A two-sample T-test for unequal variances allowed determining how significant these differences are. The claim that POEPMcreate helps generate a larger quantity of requirement ideas than does EPMcreate is statistically significant at the $\alpha = 0.05$ level, with $P = 0.0087$. The claim that EPMcreate helps generate a larger quantity of requirement ideas than does brainstorming is statistically significant not, at the $\alpha = 0.05$ level, but only at the $\alpha = 0.10$ level, with $P = 0.088$. It so happens that the claim that POEPMcreate helps generate a larger quantity of requirement ideas than does brainstorming is statistically significant at the $\alpha = 0.06$ level, with $P = 0.053$.

4.2 Evaluation of the Quality of the Requirement Ideas

The classification of the generated requirement ideas into the four ranks was carried out as described in Section 3.2 by two domain experts. The Pearson test showed that the correlation between the two experts' classifications was r = 0.68, a strongly positive correlation that is significant at the .01 level. Therefore, it is reasonable to use each expert's classification and ranking of the quality of the generated ideas.

Figure 3 shows the number of ideas generated by each group that received each quality classification, according to the first expert, and Figure 4 shows the number of ideas generated by each group that received each quality classification, according to the second expert. In the following analysis, a *new* idea is one classified with a 1 or 2. These data show that:

- Of the 74 and 76 ideas the two POEPMcreate groups generated, the average expert classified 70.5 (95%) and 70.5 (90.3%) of them, respectively, as new.
- Of the 63 and 60 ideas the two EPMcreate groups generated, the average expert classified 62 (98%) and 56 (93%) of them, respectively, as new.
- Of the 47 and 36 ideas the two brainstorming groups generated, the average expert classified 45 (96%) and 32 (88%) of them, respectively, as new.

Also these data show differences between the CETs.

The same T-test allowed determining how significant these differences are. The claim that POEPMcreate helps generate more high quality requirement ideas than does EPMcreate is statistically significant at the $\alpha = 0.085$ level, with $P = 0.081$. The claim that EPMcreate helps generate more high quality requirement ideas than does brainstorming is statistically significant at the $\alpha = 0.12$ level, with $P = 0.11$. Again, it so happens that the claim that POEPMcreate helps generate more high quality requirement ideas than does brainstorming is statistically significant at the $\alpha = 0.07$ level, with $P = 0.064$.

4.3 Analysis of Corroboratory Data

During the sessions, the subjects volunteered observations and opinions about the CETs they were using and that they saw others using. More than one POEPMcreate user said that the POEMcreate is easy to apply and the four foci were helpful. One EPMcreate user complained of having to rush and to change focus 16 times. More than one POEPMcreate and EPMcreate user said that they felt productive and that they felt that they had not missed anything as they might have in brainstorming. (Recall that every subject knew about brainstorming as a CET.) More than one brainstorming user said that they felt unfocused, that they jumped among ideas, and that they might have missed ideas. These observations and opinions corroborate the results of the data and suggest reasons that EPMcreate and its optimization work as well as they do.

Indeed, one might ask "Why is EPMcreate more effective in helping to generate requirement ideas than brainstorming?" and "Why is POEPMcreate more effective in helping to generate requirement ideas than EPMcreate?" After having observed subjects using these CETs, we suspect that any space-covering variation of EPMcreate is more effective than brainstorming, because EPMcreate gives a way to systematically visit the entire idea space. With brainstorming, one may wander aimlessly, overvisiting some

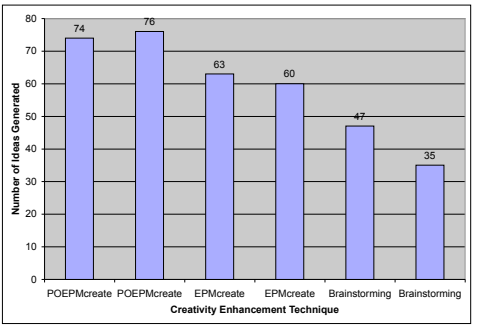


Fig. 2. Graph of the Requirement Ideas Generated by the Groups

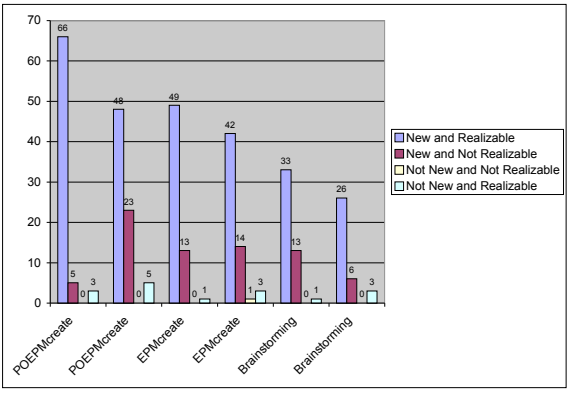


Fig. 3. First Expert's Classifications of Requirement Ideas Generated by Groups

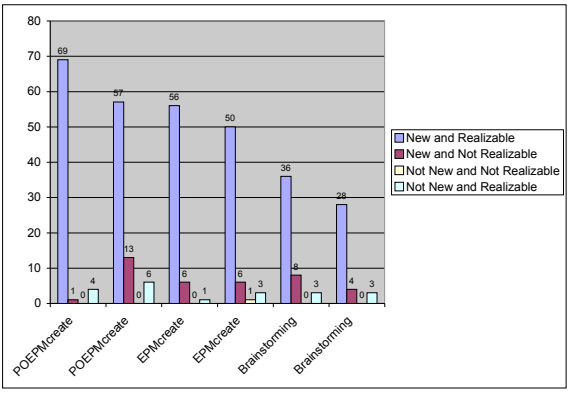


Fig. 4. Second Expert's Classifications of Requirement Ideas Generated by Groups

parts of the space and undervisiting other parts of the space. We suspect that POEPMcreate is more effective than EPMcreate because it gives a way to visit the entire space in fewer steps, with fewer mind shifts between the steps.

4.4 Summary of Analysis

In spite of the small number of data points, that might argue against significance, the data about both the quantity and the quality of the ideas generated do produce results that, based on a two-sample T-test for unequal variances, are statistically significant at various levels ranging from 0.05 to 0.12. The statistical results are corroborated by the observations and opinions of the subjects. These results thus indicate that POEPMcreate helps to generate more and better requirement ideas than EPMcreate does and that EPMcreate helps to generate more and better requirement ideas than brainstorming does. We therefore conclude that Hypotheses H1 and H2 are supported and that the experiment should be replicated.

5 Threats to Validity and Limitations

The results, despite being somewhat statistically significant, do suffer from the small numbers of subjects, groups, and thus data points involved in the experiment. It was difficult to convince students in our School of Computer Science to be subjects. Nevertheless, the results are so promising that we are planning to conduct more experiments. We will advertise for subjects in the whole university and will use smaller groups. These differences will give us a chance to see if the results are independent of the subject's major field and of the size of the groups.

Regardless, the small number of data points causes the threat of a so-called Type I error, that of accepting a non-null hypothesis, making a positive claim, when it should be rejected. Even if the data yield statistically significant results, the small number of data points increases the probability that the positive observations were random false positives. The only remedy for this threat is to have more data points or to replicate the experiment, which we are already planning to do.

Construct validity is the extent to which the experiment and its various measures test and measure what they claim to test and measure. Certainly, the groups were trying to be creative in their idea generation. Counting of raw ideas is correct, because as mentioned, at least one of the CETs compared has as a principal goal the generation of as many ideas as possible. The method to evaluate the quality of an idea, determining its novelty and its usefulness, is based squarely on an accepted definition of creativity, that it generates novel and useful ideas.

The shakiest measure used in the experiment is the Williams test of native creativity. With any psychometric test, such as the Williams test and the standard IQ tests, there is always the question of whether the test measures what its designers say it measures. The seminal paper describing the test discusses this issue [16], and the test seems to be accepted in the academic psychometric testing field [18]. The original test was designed for testing children, and the test seems to be used in U.S. schools to identify gifted and talented students [19]. We modified the test to be for adults attending a university

or working [9,17]. Each of the authors has examined the test and has determined for him- or herself that the test does examine at least something related to creativity if not native creativity itself. Finally, the same modified-for-adults Williams test, in Italian and English versions, has been used in all of our past experiments about CETs and will be used in all of our future experiments about CETs. Therefore, even if the test does not measure native creativity exactly or fully, the same error is made in all our experiments so that the results of all of these experiments should be comparable.

Internal validity is whether one can conclude the causal relationship that is being tested by the experiment. In this case, we are claiming that the differences in CETs caused the observed differences in the quantity and quality of the requirement ideas generated by use of the CETs. We know from being in the room with the groups that each group was actively using its assigned CET while it was generated its ideas. We carefully assigned subjects to the groups so that the groups were balanced in all personal factors, especially native creativity, that we thought might influence the subjects' abilities to generate requirement ideas. Therefore, we believe that the only factor that can account for the differences in the number of ideas is the CET being used by the groups. The opinions volunteered by the subjects during the sessions corroborate this belief.

External validity is whether the results can be generalized to other cases, with different kinds of subjects, with different kinds of CBS. Certainly the small number of data points stands in the way of generalization.

One threat to external validity is the use of students as subjects instead of requirements elicitation or software development professionals. However, our student subjects had all studied at least a few courses in computer science and software engineering. Moreover, each group had at least one subject with professional experience in computing. One could argue that the subjects were equivalent to young professionals, each at an early stage in his or her career [20].

Another threat to external validity is the particular choice of the types of stakeholders whose viewpoints were used by EPMcreate and POEPMcreate sessions. Would other choices, e.g., of teachers, work as well?

Yet another threat to external validity is the single Web site as the CBS for which to generate requirement ideas. Would a different Web site or even a different kind of CBS inhibit the effectiveness of any CET?

In any case, our plans for future experiments, to use different kinds of subjects, different sized groups, different stakeholder viewpoints, and different CBSs for which to generate requirement ideas, address these threats to external validity.

These threats limit the strength of the conclusion of support for the hypotheses and dictate the necessity to replicate the experiments.

6 Conclusions

This paper has described an experiment to compare the effectiveness of three CETs, EPMcreate, POEPMcreate, and brainstorming. The experiment tested two hypotheses that say that POEPMcreate is more effective in helping to generate new requirement ideas than EPMcreate, which is in turn more effective in helping to generate new

requirement ideas than brainstorming. The data from the experiment support both hypotheses, albeit not with uniformly high significance, due to the low number of subjects participating in the experiment. However, the support is strong enough that it is worth conducting more experiments to test these hypotheses, with more subjects and different CBSs about which to generate requirement ideas. Should you want to conduct these experiments, please avail yourself of the experimental materials we used [21].

It is necessary also to compare POEPMcreate with CETs other than EPMcreate and brainstorming. We suggest also to evaluate the effectiveness of other optimizations of EPMcreate and of other orderings of the steps of EPMcreate, POEPMcreate, and the other optimizations.

Mich, Berry, and Alzetta [17] have compared the effectiveness of EPMcreate applied by individuals to the effectiveness of EPMcreate applied by groups. It will be interesting to do a similar comparison for POEPMcreate and other optimizations that prove to be effective.

Finally, recall that Hypothesis H1 was stronger than needed. All that was required to satisfy us is that POEPMcreate be at least as effective than EPMcreate. While the support for H1 is not as strong as desired, the support for a logical union of H1 and its null hypothesis would be stronger. As an optimization, POEPMcreate is easier to apply and easier to teach than EPMcreate. POEPMcreate's fewer steps means that either it requires less time to use or there is more more time in each step for idea generation. POEPMcreate's fewer steps means that less time is wasted shifting the user's mental focus.

Acknowledgments

The authors thank William Berry for his advice on matters of statistical significance. They thank the referees and shepherds for their comments, and in particular, they thank Sam Fricker for his persistence and his conciseness improving suggestions. Victoria Sakhnini's and Luisa Mich's work was supported in parts by a Cheriton School of Computer Science addendum to the same Canadian NSERC–Scotia Bank Industrial Research Chair that is supporting Daniel Berry. Daniel Berry's work was supported in parts by a Canadian NSERC grant NSERC-RGPIN227055-00 and by a Canadian NSERC–Scotia Bank Industrial Research Chair NSERC-IRCPJ365473-05.

References

1. Gause, D., Weinberg, G.: Exploring Requirements: Quality Before Design. Dorset House, New York (1989)
2. Maiden, N., Gizikis, A., Robertson, S.: Provoking creativity: Imagine what your requirements could be like. IEEE Software 21, 68–75 (2004)
3. Nguyen, L., Shanks, G.: A framework for understanding creativity in requirements engineering. J. Information & Software Technology 51, 655–662 (2009)
4. Osborn, A.: Applied Imagination. Charles Scribner's, New York (1953)
5. Bono, E.D.: Six Thinking Hats. Viking, London (1985)
6. Bono, E.D.: Serious Creativity: Using the Power of Lateral Thinking to Create New Ideas. Harper Collins, London (1993)

7. Aurum, A., Martin, E.: Requirements elicitation using solo brainstorming. In: Proc. 3rd Australian Conf. on Requirements Engineering, pp. 29–37. Deakin University, Australia (1998)
8. Jones, S., Lynch, P., Maiden, N., Lindstaedt, S.: Use and influence of creative ideas and requirements for a work-integrated learning system. In: Proc. 16th IEEE International Requirements Engineering Conference, RE 2008, pp. 289–294. IEEE Computer Society, Los Alamitos (2008)
9. Mich, L., Anesi, C., Berry, D.M.: Applying a pragmatics-based creativity-fostering technique to requirements elicitation. Requirements Engineering J. 10, 262–274 (2005)
10. Mich, L., Berry, D.M., Franch, M.: Classifying web-application requirement ideas generated using creativity fostering techniques according to a quality model for web applications. In: Proc. 12th Int. Workshop Requirements Engineering: Foundation for Software Quality, REFSQ 2006 (2006)
11. Lefons, E., Pazienza, M.T., Silvestri, A., Tangorra, F., Corfiati, L., De Giacomo, P.: An algebraic model for systems of psychically interacting subjects. In: Dubuisson, O. (ed.) Proc. IFAC Workshop Information & Systems, Compiegne, France, pp. 155–163 (1977)
12. De Giacomo, P.: Mente e Creatività: Il Modello Pragmatico Elementare Quale Strumento per Sviuppare la Creatività in Campo Medico, Psicologico e Manageriale. In: Franco Angeli, Milano, Italy (1995) (in Italian)
13. Leffingwell, D., Widrig, D.: Managing Software Requirements: a Unified Approach, 5th edn. Addison-Wesley, Boston (1999)
14. Telem, M.: Information requirements specification I & II: Brainstorming collective decision-making approach. Information Processing Management 24, 549–557, 559–566 (1988)
15. Administrator: Sir John A MacDonald High School Web Site (Viewed November 16-20, 2009), http://sja.ednet.ns.ca/index.html
16. Williams, F., Taylor, C.W.: Instructional media and creativity. In: Proc. 6th Utah Creativity Research Conf., New York, NY, USA. Wiley, Chichester (1966)
17. Mich, L., Berry, D.M., Alzetta, A.: Individual and end-user application of the epmcreate creativity enhancement technique to website requirements elicitation. Technical report, School of Computer Science, University of Waterloo (2009), http://se.uwaterloo.ca/∼dberry/FTP_SITE/tech.reports/MichBerryAlzetta.pdf
18. Dow, G.: Creativity Test: Creativity Assessment Packet (Williams, 1980), R546 Instructional Strategies for Thinking, Collaboration, and Motivation, AKA: Best of Bonk on the Web (BOBWEB). Technical report, Indiana University (Viewed March 7, 2010)
19. West Side School District: Gifted and Talented Program. Technical report, West Side Public Schools, Higden, AR, USA (Viewed March 7, 2010)
20. Berander, P.: Using students as subjects in requirements prioritization. In: Proceedings of the International Symposium on Empirical Software Engineering (ISESE 2004), pp. 167–176. IEEE Computer Society, Los Alamitos (2004)
21. Sakhnini, V., Berry, D., Mich, L.: Materials for Comparing POEPMcreate, EPMcreate, and Brainstorming. Technical report, School of Computer Science, University of Waterloo (Viewed March 7, 2010), http://se.uwaterloo.ca/∼dberry/FTP_SITE/software.distribution/EPMcreateExperimentMaterials/

Towards Multi-view Feature-Based Configuration

Arnaud Hubaux[1], Patrick Heymans[1], Pierre-Yves Schobbens[1], and Dirk Deridder[2]

[1] PReCISE Research Centre, Faculty of Computer Science, University of Namur
Namur, Belgium
{ahu,phe,pys}@info.fundp.ac.be
[2] Software Languages Lab, Vrije Universiteit Brussel,
Brussels, Belgium
Dirk.Deridder@vub.ac.be

Abstract. [Context & motivation] Variability models, feature diagrams ahead, have become commonplace in software product line engineering as a means to document variability early in the lifecycle. Over the years though, their application span has been extended to aid stakeholders in the configuration of software products. **[Question/problem]** However, current feature-based configuration techniques hardly support the tailoring of configuration views to the profiles of heterogeneous stakeholders. **[Principal ideas/results]** In this paper, we introduce a lightweight mechanism to leverage multidimensional separation of concerns in feature-based configuration. **[Contribution]** We propose a technique to specify concerns in feature diagrams and to build automatically concern-specific configuration views, which come with three alternative visualisations.

1 Introduction

An increasing number of software developments adopt the paradigm of software product line engineering (SPLE) [1]. The goal of SPLE is to rationalise the development of families of similar software products. A key idea is to institutionalise reuse throughout the development process to accomplish economies of scale.

SPLE is a very active research area at the crossroads between many software development related disciplines, including requirements engineering (RE). An important research topic in SPLE and RE is feature diagrams (FDs) [2,3]. FDs are a simple graphical formalism whose main purpose is to document variability in terms of *features*, i.e. high-level descriptions of the capabilities of the reusable artefacts.

FDs have been given a formal semantics [3], which opened the way for safe and efficient automation of various, otherwise error-prone and tedious, tasks including consistency checking [4,5], decision propagation [4] and process control [6,7,8]. A repertoire of such automations can be found in [9]. The kind of automation that we focus on in this paper is feature-based configuration (FBC). FBC is an interactive process during which one or more stakeholders select and discard features to build specific products. FBC is one of the principal means to elicit product requirements in SPLE. In practice, there can be thousands of features whose legal combinations are governed by many and often complex rules [4]. It is thus of crucial importance to be able to simplify and automate the decision-making process as much as possible.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 106–112, 2010.

Two challenges that FBC techniques fail to address in a satisfactory way are (1) *tailoring the configuration environment* according to the stakeholder's profile (knowledge, role, preferences...) and (2) *managing the complexity* resulting from the size of the FD.

In this paper, we outline a solution strategy to address those two challenges. We do so by extending FDs with multiple views that can be used to automatically build FD visualisations. A view is a streamlined representation of a FD that has been tailored for a specific stakeholder, task, or, to generalize, a combination of such elements—which we call a *concern*. Views facilitate configuration in that they only focus on those parts of the FD that are relevant for a given concern. Using multiple views is thus a way to achieve *separation of concerns* (SoC) in FDs. SoC helps making FD-related tasks less complex by letting stakeholders concentrate on the parts that are relevant to them while hiding the others. Further tailoring of the visualisations is suggested through the selection of three alternative visualisations: (1) "greyed out", (2) "pruned" and (3) "collapsed".

In the rest of this paper, we elaborate on these ideas. Section 2 introduces FDs. A motivating example is given in Section 3. Section 4 presents our basic strategy for constructing views.

2 Feature Diagram

Schobbens *et al.* [3] defined a generic formal semantics for a wide range of FD dialects. We only recall the basic concepts. In essence, a FD d is a hierarchy of features (typically a tree) topped by a root feature. Each feature has a cardinality $\langle i..j \rangle$ attached to it, where i (resp. j) is the minimum (resp. maximum) number of children (i.e. features at the level below) required in a product (aka configuration). For convenience, common cardinalities are denoted by Boolean operators, as shown in Table 1. Additional constraints that crosscut the tree can also be added and are defined, without loss of generality, as a conjunction of Boolean formulae. The semantics of a FD is the set of products. The full syntax and semantics as well as benefits, limitations and applications of FDs are extensively discussed elsewhere [3,9].

FBC tools use FDs to pilot the configuration of customisable products. These tools usually render FDs in an *explorer-view* style [10,4], as in the upper part of Table 1. The tick boxes in front of features are used to capture decisions, i.e. whether the features are selected or not. We now illustrate this more concretely with a motivating example.

Table 1. FD decomposition operators

Concrete syntax	f ☐ g ☐ h	f ☐ g ☐ h	f ☐ g ☐ h	f ⟨i..j⟩ ☐ g ☐ h	☐ g
Boolean operator	and: $\wedge$	or: $\vee$	xor: $\oplus$	non standard	optional
Cardinality	$\langle n..n \rangle$	$\langle 1..n \rangle$	$\langle 1..1 \rangle$	$\langle i..j \rangle$	$\langle 0..1 \rangle$

3 Motivating Example

Spacebel is a Belgian software company developing software for the aerospace industry. We collaborate with Spacebel on the development of a SPL for flight-grade libraries implementing the CSSDS File Delivery Protocol (CFDP) [8]. CFDP is a file transfer protocol designed for space requirements, such as long transmission delays and specific hardware characterised by stringent resource limitations. Spacebel built a SPL of CFDP libraries, where each library can be tailored to the needs of a specific space mission.

The FD of the CFDP SPL counts 80 features, has a maximal depth of four and contains ten additional constraints. A simplified excerpt of this FD appears in the upper part of Figure 1[1]. The principal features provide the capability to send (*Send*) and receive (*Receive*) files. The *Extended* feature allows a device to send and receive packets via other devices. The *Reboot* feature allows the protocol to resume transfers safely after a sudden system reboot. PUS stands for Packet Utilisation Standard, part of the ESA

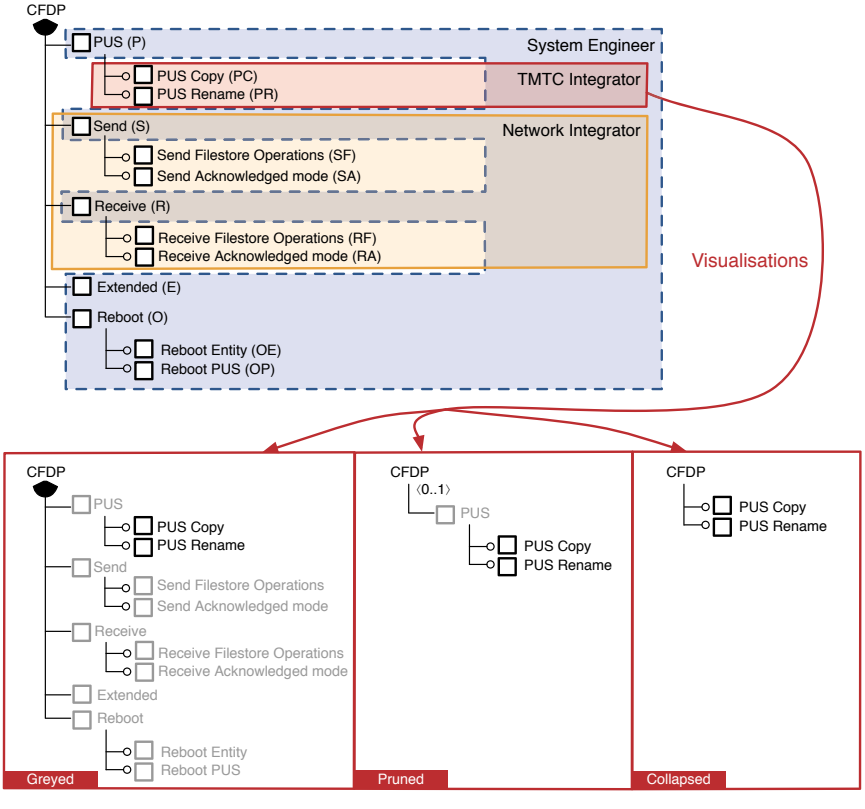


Fig. 1. FD of the CFDP with three alternative visualisations for the view of the TMTC integrator

[1] An online version designed with SPLOT, an open source web-based FBC tool, is available at `http://www.splot-research.org/`.

standard for transport of telemetry and telecommand data (TMTC). The *PUS* feature implements the CFDP related services of this standard.

CFDP typically handles four different stakeholder profiles. *Spacebel* decides which features are mature enough for the mission, while leaving as much variability as possible. The *system engineer* makes initial high-level choices and passes the task of refining these choices on to the *network integrator* and the *TMTC integrator* who handle the technical aspects of the CFDP. The configuration options of interest for each of these profiles are thus different and limited in scope.

A major problem is that access rights to these configuration options are currently informally defined and FDs offer no way to do so. In the absence of clear access specifications, a simplistic policy has been implemented: all profiles have access to all configuration options. A reported consequence is that sometimes the system engineer does not have sufficient knowledge to fully understand low-level options and make decisions. The results were incorrect settings, e.g., inappropriate CPU consumption or excessive use of memory for a given hardware. Similarly, the integrators were not aware of general decisions and could make inconsistent choices wrt. the missions' goals.

The changing context also demands flexible definitions of access policies. For instance, there can be variations in the access rights (e.g., the integrators are granted access to more features) or stakeholder profiles (e.g. a dedicated *File System integrator* might be needed in some projects).

This situation provided the initial motivation for the solution outlined in this paper. However, as we will see, the solution is applicable to a wider variety of problems than the sole definition of configuration access rights. Its ambition is to extend FDs with support for multiple perspectives.

4 Multi-view Feature Diagrams

Solving the problem described in the previous section requires being able to specify which parts of the FD are configurable by whom. This can be achieved easily by augmenting the FD with a set V of views, each of which consists of a set of features. Each view $v_i \in V$ contains some features of the FD. A view can be defined for any concern that requires only partial knowledge of the FD. Also, as a general policy, we consider that the root is part of each view.

View specification. There are essentially two ways of specifying views. The most obvious is to enumerate, for each view, the features that appear in it, or equivalently, to tag each feature of the FD with the names of the views it belongs too. These are *extensional definitions*, which might be very time-consuming and error-prone if the FD is too big and there is no appropriate tool support. A natural alternative is thus to provide a language for *intensional definitions* of views that takes advantage of the FD's tree structure to avoid lengthy enumerations. A simple query language like XPath would be a good candidate to define views and retrieve the corresponding features.

In Figure 1, views have been illustrated by coloured areas. The first area (blue) consists of the high-level features that should be accessible to the system engineer. The last

two areas (red and orange) respectively contain the technical features that should be accessible to the TMTC and network integrators.

View coverage. An important property to be guaranteed by a FBC system is that all configuration questions be eventually answered [7], i.e. that a decision be made for each feature of the FD. A *sufficient condition* is to check that all the features in the FD are in the views of V. The FD of Figure 1 fulfils that condition. But this is not necessary since some decisions can usually be deduced from others.

A *necessary and sufficient condition* can be defined using the notion of propositional defineability [11]. We need to ensure that the decisions on the features that do not appear in any view can be inferred from (are *propositionally defined by*) the decisions made on the features that are part of the view. This can be achieved by translating the FD into an equivalent propositional formula and apply the algorithm described in [11]. Features that do not belong to any view and that do not satisfy the above condition will have to be added to existing views, or new views will have to be created to configure them.

View interactions. Another important property of FBC is that it should always lead to valid configurations [7]. In our case, doing the configuration through multiple views is not a problem *per se*. This is because, although stakeholders only have partial views, the FBC system knows the whole FD and is thus capable of propagating the choices made in one view to the others. However, problems can arise when the selection of a feature in one view depends on the selection of another feature in another view. If overriding of decisions across views is not allowed, then we must introduce some form of conflict resolution mechanisms. This is a complex issue for which various strategies can be elaborated. One is to introduce priorities on views [12]. Another one is to constrain the order in which views are configured [8].

Visualisation. Views are abstract entities. To be effectively used during FBC, they need to be made concrete, i.e. visual. We call a visual representation of a view a *visualisation*. The goal of a visualisation is to strike a balance between (1) showing only features that belong to a concern and (2) including features that are *not* in the the concern but that allow the user to make informed decisions. For instance, the *PUS copy* feature is in the view of the TMTC integrator, but its parent feature *PUS* is not: How will that influence the decision making process? To tackle this problem, we see three visualisation alternatives with different levels of details (see lower part of Figure 1).

The *greyed* visualisation is a mere copy of the original FD in which the features that do not belong to the view are greyed out (e.g. P, S, SF and SA). Greyed out features are only displayed but cannot be manually selected/deselected. In the *pruned* visualisation, features that are not in the view are pruned (e.g. S, SF and SA) unless they appear on a path between a feature in the view and the root, in which case they are greyed out (e.g. P). Pruning can have an impact on cardinalities. As shown in Figure 1, the cardinality of *CFDP* is $\langle 0..1 \rangle$ whereas it is $\langle 1..5 \rangle$ (or-decomposition) in the FD. It has to be recomputed to ensure the consistency of the FD. In the *collapsed* visualisation, all the features that do not belong to the view are pruned. A feature in the view whose parent or ancestors are pruned is connected to the closest ancestor

that is still in the view. If no ancestor is in the view, the feature is directly connected to the root (e.g. PC and PR). Similarly, cardinalities have to be recomputed for consistency reasons.

5 Conclusion

In this paper, we have outlined an approach to specify views on feature diagrams in order to facilitate feature-based configuration, one of the main techniques to define product requirements in software product lines. Three alternative visualisations were proposed, each offering different levels of detail. This work was motivated by an ongoing collaboration with the developers of Spacebel, a Belgian software company developing software for the aerospace industry. A preliminary evaluation with the developers of an open source web-based meeting management system is also in progress [13].

A number of future work can be envisioned. First, a more thorough evaluation should be carried out. Second, we will have to address the problem of conflictual configuration decisions across views. Third, the formalisation needs to be refined and the implementation to be pursued. Currently, we only have standalone algorithms implementing our transformations. The rest of our approach needs to be developed, integrated in a feature modelling and configuration environment, and properly validated.

Acknowledgements

This work is sponsored by the Interuniversity Attraction Poles Programme of the Belgian State, Belgian Science Policy, under the MoVES project.

References

1. Pohl, K., Bockle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (July 2005)
2. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, SEI, Carnegie Mellon University (November 1990)
3. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature Diagrams: A Survey and A Formal Semantics. In: RE 2006, pp. 139–148 (September 2006)
4. Mendonça, M.: Efficient Reasoning Techniques for Large Scale Feature Models. PhD thesis, University of Waterloo (2009)
5. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
6. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multi-level configuration of feature models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
7. Classen, A., Hubaux, A., Heymans, P.: A formal semantics for multi-level staged configuration. In: VaMoS 2009. University of Duisburg-Essen (January 2009)
8. Hubaux, A., Classen, A., Heymans, P.: Formal modelling of feature configuration workflow. In: SPLC 2009, San Francisco, CA, USA (2009)

9. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. IST (2010) (preprint)
10. pure-systems GmbH: Variant management with pure: variants. Technical White Paper (2006), http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf
11. Lang, J., Marquis, P.: On propositional definability. Artificial Intelligence 172(8-9), 991–1017 (2008)
12. Zhao, H., Zhang, W., Mei, H.: Multi-view based customization of feature models. Journal of Frontiers of Computer Science and Technology 2(3), 260–273 (2008)
13. Hubaux, A., Heymans, P., Schobbens, P.Y.: Supporting mulitple perspectives in feature-based configuration: Foundations. Technical Report P-CS-TR MPFD-000001, PReCISE Research Centre, Univ. of Namur (2010), http://www.fundp.ac.be/pdf/publications/69578.pdf

Evaluation of a Method for Proactively Managing the Evolving Scope of a Software Product Line

Karina Villela, Jörg Dörr, and Isabel John

Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{Karina.Villela,Joerg.Doerr,Isabel.John}@iese.fraunhofer.de

Abstract. [Context and motivation] PLEvo-Scoping is a method intended to help Product Line (PL) scoping teams anticipate emergent features and distinguish unstable from stable features, with the aim of preparing their PL for likely future adaptation needs. **[Question/problem]** This paper describes a quasi-experiment performed to characterize PLEvo-Scoping in terms of adequacy and feasibility. **[Principal ideas/results]** This quasi-experiment was performed by two scoping teams in charge of scoping the same PL, where one scoping team applied first an existing PL scoping approach and then PLEvo-Scoping, while the other scoping team interweaved activities from both. The two approaches achieved similar results: The method could be applied in just one day, and it was considered adequate and feasible. **[Contribution]** Ideas on how to improve the method and its tool support have been obtained, and similar results are expected from other professionals facing the problem of evolution-centered PL scoping. However, further empirical studies should be performed.

Keywords: Quasi-experiment, Empirical Study, Product Line Scoping, Requirements Volatility Analysis, Product Line Evolution Planning.

1 Introduction

Product Line (PL) Engineering is a software development approach that aims at exploiting commonalities and predicted variabilities among software products that strongly overlap in terms of functionality [1,2]. According to Knauber and Succi [3], PLs are already intended to capture the evolution of software products and to last for a fairly long time. In this context, one of the most important aspects to consider is the ability of PLs themselves to evolve and change [3]. However, Savolainen and Kuusela [4] emphasize that any given design can only handle a limited number of different kinds of changes and, therefore, it is crucial to predict what kind of changes will be required during the lifespan of a PL.

Current PL engineering methods [1,2,5] address pre-planned and more straightforward proactive changes across products or different versions of products. They do not support the prediction of the not-so-straightforward future changes in products and features, which are often triggered by change requests from inside or outside the organization (such as a change due to the decision of a technology provider to

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 113–127, 2010.

discontinue the production of a hardware component). Therefore, a good commonality and variability analysis is not enough [6].

PL scoping is the PL requirements engineering step in charge of deciding in which parts of an organization's products systematic reuse is economically useful and should thus be supported by a PL infrastructure [7,8]. When focusing on this phase, it is customary to use the identified requirements that are outside the current scope as candidates for future addition as well as to use those requirements to evaluate the design of the current PL. However, this does not go as far as an explicit volatility analysis.

The method PLEvo-Scoping (Product Line Evolution Support at Scoping) was defined to complement and extend PL scoping approaches by helping the PL scoping team anticipate emergent features and distinguish unstable from stable PL features. The aim is to prepare the PL for likely future adaptation needs by planning for changes beforehand. Thus, the PL scoping approach can keep its focus on current, planned, and potential PL products and features, while PLEvo-Scoping addresses the prediction and planning for the evolution of these products and features, while also contributing to the discovery of further innovative features.

This article describes a quasi-experiment performed to characterize the adequacy and feasibility of PLEvo-Scoping with a twofold purpose: 1) obtaining feedback from PL practitioners on how to improve the method; and 2) providing first empirical data to help PL organizations decide about trying out the method. Two scoping teams (each composed of three professionals) had the task of scoping the same PL by using PLEvo-Scoping together with an existing PL scoping approach, but with two different arrangements of sub-tasks. We believe this quasi-experiment has been a relevant step towards the evaluation of the method against other potential approaches, because any PL organization is only willing to test a new method if there is initial evidence that the benefits to be obtained are worth the effort required to apply it. It is not the goal of this paper to describe the state of the art related to PLEvo-Scoping or the method itself in detail; this information can be found in [9] and [10].

In the reported quasi-experiment, PLEvo-Scoping was applied in the Ambient Assisted Living (AAL) domain for the second time. Products in this domain support elderly people in staying at home longer instead of moving to a nursing home. However, the two applications of the method in this domain were completely different. The first one [9] can be considered an assertion study [11], in which one of the authors applied the method herself after studying the application domain, while domain experts just validated the method results, indicating the necessary corrections. The object was the specific AAL services to be provided by an AAL demonstrator. Concerning the quasi-experiment, the method was exclusively applied by the two PL scoping teams and the object was the AAL platform PL. An AAL platform is an architectural component that provides infrastructure services on top of which the specific AAL services are provided.

In the next sections, we provide an overview of PLEvo-Scoping (Section 2) and briefly discuss how it interfaces with existing PL scoping approaches (Section 3), so that the quasi-experiment can be understood better. Section 4 reports the quasi-experiment and is organized as follows: definition, planning, operation, data analysis, and comments and interpretation of the results. The last section contains our conclusions and the next stages of this research.

2 Method Overview

PLEvo-Scoping consists of four steps to be carried out by the PL scoping team, which is generally composed of people with the following roles [8]: scoping expert, PL manager, and domain expert, the latter with either the technical or the market point of view.

The first step is *Preparation for Volatility Analysis*, which establishes the basis for the volatility analysis and is made up of the following activities:

- *Activity 1:* Establish the timeframe that restricts the current volatility analysis, and
- *Activity 2:* Identify/update the types of system components that are generally involved in the assembly of the planned PL products.

The second step is called *Environment Change Anticipation* and has the purpose of identifying and characterizing facts that may take place in the PL's environment within the pre-established timeframe, and that may allow or require adaptations in the PL. This step comprises the following activities:

- *Activity 3:* Identify the actors that play a role in the PL's environment and who give rise to or realize facts that may affect the PL,
- *Activity 4:* Identify and characterize facts that may be caused or realized by the identified actors and have the potential for changing the PL's environment,
- *Activity 5:* Verify the perspective of new actors playing a part in the PL's environment within the volatility analysis timeframe and characterize how these actors may change such an environment, and
- *Activity 6:* Classify the previously characterized facts according to their relevance, in order to decide whether and when they should have their impact in terms of adaptation needs analyzed.

The next step is called *Change Impact Analysis*. Its purpose is to analyze the impact of the identified facts on the PL and consists of:

- *Activity 7:* Identify the adaptation needs that may be allowed or required in the PL as a consequence of the previously identified facts,
- *Activity 8:* Characterize the adaptation needs by identifying the PL features to be affected by them, and by estimating their business impact, technical penalty, and technical risk, and
- *Activity 9:* Classify the adaptation needs according to relevance, in order to decide whether and when the inclusion of an adaptation need should be planned.

Once the most relevant adaptation needs have been selected, it is time for *PL Evolution Planning*. The idea is to establish when and how relevant adaptation needs are expected to be introduced into the PL, and prepare it for accommodating the adaptation needs beforehand. The activities that make up this step are:

- *Activity 10:* Determine when and in which products relevant adaptations are expected to be introduced, which gives rise to the *PL Evolution Map*,
- *Activity 11:* Analyze the alternative solutions for dealing with relevant adaptation needs, in terms of effort, cost, effectiveness, and strategic alignment,
- *Activity 12:* Select the best alternatives for dealing with the adaptation needs, and
- *Activity 13:* Revise the *PL Evolution Map* in order to adjust it to the alternative solutions selected, if necessary.

Procedural descriptions, checklists, guidelines, and templates of documents are provided to support the PL scoping team in carrying out each activity of the method. In addition, optional activities have been identified in order to address the different levels of experience in the domain that different teams may have, and also to take into account time restrictions, by providing a light-weight variant of the method. Most optional activities use systematic reasoning to identify and characterize facts and adaptation needs that otherwise would have to be identified from scratch. Therefore, they are very important in terms of the completeness of the method results, especially when the PL scoping team is not very experienced in the application domain. The other optional activities aim at making the method scalable when too many facts or adaptation needs are identified.

After applying the method, the most relevant adaptation needs likely to be allowed or required within the volatility analysis timeframe are expected to be identified and included in a plan for PL evolution. Excerpts from output documents can be found in [9] and [10].

3 Integration into the Scoping Process

PLEvo-Scoping is expected to be carried out after or in parallel to the standard PL scoping process, provided the mandatory method inputs are available (*Description of Products* and *List of PL Features*). Broadly speaking, the method takes those mandatory inputs together with relevant information for reasoning on requirements volatility, and helps to identify a set of adaptations likely to be requested or allowed in an established future timeframe, pointing out on a map how the PL features will be affected by these adaptations. New features may arise; others may become obsolete; it may be revealed that a feature will be affected by many potential future adaptations and therefore is an unstable feature; or it may be revealed that an adaptation will affect many different features and thus requires special attention. Therefore, the *List of PL Features* can be improved, by including the newly identified features, characterizing the PL features according to their volatility, and changing the priority of some PL features based on their volatility or relevance for a certain adaptation need. The *Product Release Plan*, which is not a mandatory but a much desired method input, is augmented with the indication of when and in which products relevant adaptation needs are expected to be introduced. Furthermore, the analysis of alternative solutions for dealing with the adaptation needs may affect the prioritization of assets to be built for the PL and will indicate what kind of evolvability should be built into them.

PuLSE-Eco [7,8] is the technical component of the PuLSE® methodology [5] in charge of supporting PL scoping. This technical component, which is the best known and best documented scoping approach so far, is used below to illustrate the integration between PLEvo-Scoping and a concrete PL scoping process, because this approach was used in the quasi-experiment that will be described in the next sections.

PuLSE-Eco's generic process comprises three phases: Product Line Mapping, Domain Potential Assessment, and Reuse Infrastructure Scoping (see the top part of Figure 1). During the *Product Line Mapping* phase, the products and features of a PL are identified and the distribution of features to products is established. The goal is to provide an overview of the PL. During the *Domain Potential Assessment* phase,

sub-domains are analyzed according to assessment dimensions in order to support the decision on whether to include these sub-domains in the PL infrastructure or not. During the *Reuse Infrastructure Scoping* phase, existing and planned assets are identified and their reuse potential is quantified. The aim of this last phase is to plan the PL infrastructure and identify development needs. Figure 1 shows the main outputs of these phases and indicates in which PLEvo-Scoping activities they are requested (mandatory inputs) or can make a contribution (optional inputs). On the other hand, the outputs of PLEvo-Scoping are used to update the *(Quantified) Product Feature Matrix* and the *Product Release Plan* as already explained, once the *(Quantified) Product Feature Matrix* contains the *List of PL Features*; as well as to augment the *Domain Assessment Report* with additional information about the stability of the domains, one of the assessment dimensions proposed in PuLSE-Eco [7]. PLEvo-Scoping can also be integrated into other scoping processes, as they all provide similar results [7].

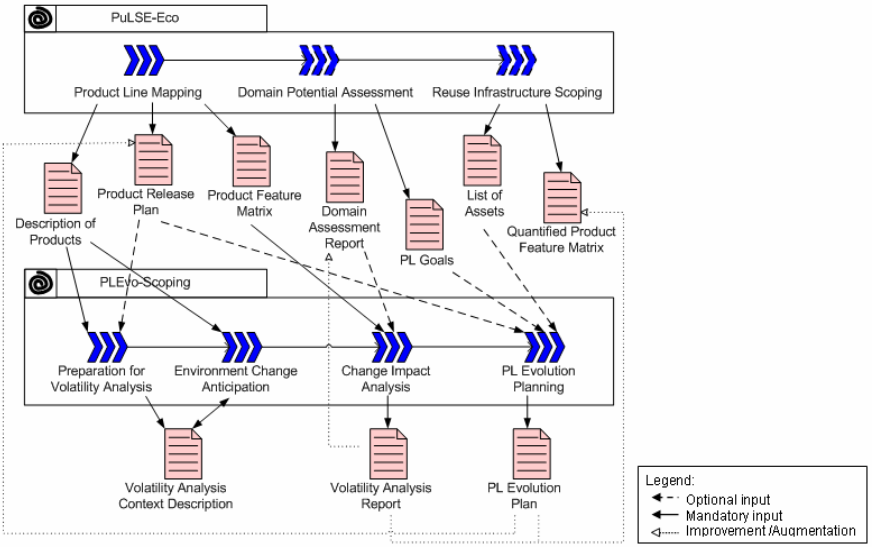


Fig. 1. Interface between a PL scoping process and PLEvo-Scoping

4 Quasi-experiment

4.1 Definition

The goal of this quasi-experiment was to characterize the adequacy and feasibility of PLEvo-Scoping, by collecting the perception of the quasi-experiment participants as well as some quantitative measures. These empirical data are expected to support PL organizations in making the decision to try out the method, which will give us the opportunity to perform further empirical studies, such as a case study in a software company. In addition, feedback provided by the quasi-experiment participants should be used to improve the method. According to the template proposed in the

Goal/Question/Metric method [12], the goal of this quasi-experiment can be represented as follows: *Analyze* the PLEvo-Scoping method *for the purpose of* characterization *with respect to* its adequacy and feasibility *from the point of view of* the researchers, PL managers, and domain experts *in the context of* Fraunhofer IESE's employees in charge of scoping an Ambient Assisted Living PL.

Concerning the context, Fraunhofer IESE has a research program on Ambient Assisted Living (AAL), which is funded by various sources in the form of independent projects. Since there are a lot of overlaps between the different projects and the AAL platform is of strategic interest for the institute, the research program leader needed scoping of an AAL platform PL, so that a PL infrastructure capable of supporting the reuse of the commonalities could be created. He also agreed to using PLEvo-Scoping to extend Fraunhofer's approach for PL scoping (PuLSE-Eco [7,8]), since a simplified version of the method had already been applied to anticipate the evolution of a single product in this domain and actually provided insightful ideas on features for the next version of the product [9]. Therefore, there was a real problem to be solved by professionals at Fraunhofer IESE, which, on the one side, strongly limited the degree of control over the quasi-experiment, but, on the other side, increased the possibility of generalizing the quasi-experiment results to other companies with PL competencies. The AAL Platform PL scoping was not carried out as part of a research project, but rather as an internal project of strategic relevance.

4.2 Planning

From the goal definition, two propositions were defined: P1) PLEvo-Scoping is adequate, and P2) PLEvo-Scoping is feasible. It was established for this quasi-experiment that PLEvo-Scoping would be considered adequate (P1) if its immediate benefits (in terms of changes in the outputs of the scoping process and information made available to guide further activities and decisions) were considered to support PL evolution, according to the judgment of the quasi-experiment participants; and feasible (P2) if the obtained benefits were worth the effort required for applying the method, which would also be judged by the quasi-experiment participants. In both cases, the quasi-experiment participants should be provided with the results of the scoping process and support their judgment with both qualitative and quantitative information.

By using the Goal/Question/Metric method [12], the quasi-experiment propositions were broken down into questions and metrics. Each metric was defined in terms of meaning, type of measure, measure scale, source, and collecting procedure [13]. The acceptance criteria were also defined during the planning of the quasi-experiment and will be presented in Subsection 4.4 (Data Analysis).

This quasi-experiment was designed to be performed in two days by two scoping teams in charge of separately scoping the AAL platform as a PL. Group 1 applied treatment 1, which consisted of first using PuLSE-Eco to conduct the scoping process, already taking into consideration the evolution concern, and then applying PLEvo-Scoping; while Group 2 applied treatment 2, which consisted of interweaving the activities of PuLSE-Eco and PLEvo-Scoping. The purpose of treatment 1 was to allow a clear distinction between the results before and after applying PLEvo-Scoping, while the purpose of treatment 2 was to avoid the confounding factor (present in treatment 1) of providing the scoping team with extra time to think about PL scope evolution after

applying PuLSE-Eco. This confounding factor would put in doubt whether the obtained benefits were a result of this extra time or a result of the application of PLEvo-Scoping. In addition, the application of PLEvo-Scoping in the two treatments aimed at strengthening the validity of the results through corroboration. Two days were allocated to each treatment due to time restrictions.

Each group consisted of three people: one PL manager and two domain experts, one with the technical point of view and the other one with the market point of view. All participants were selected by the leader of the AAL research program, taking into account their profiles and involvement in AAL research projects. Two experts on the respective two methods (PuLSE-Eco and PLEvo-Scoping) were allocated to guide the pertinent part of the scoping process in both treatments. Neither the PuLSE-Eco nor the PLEvo-Scoping expert was involved in the AAL research program, so their role in the scoping process was comparable to that of an external consultant.

As PL scoping teams are not generally composed of many people, all necessary roles were represented, and the PL scoping process was to take just a couple of days [8]; this design was realistic compared to industrial settings.

The threats to the validity of this quasi-experiment have been analyzed based on the set of threats to validity provided in [14]. We have addressed most validity threats:

- *Fishing*: Subjective classifications and measures were only provided by the participants of the quasi-experiment; two types of open questions were included; two people with no special expectations in the quasi-experiment results were involved in its data analysis.
- *Reliability of measures*: The quasi-experiment participants defined subjective measures and/or provided values for them based on objective measures; the instruments were revised by three people with different profiles (one M.Sc. student, one PL professional, and one expert in empirical studies).
- *Mono-method bias*: Both quantitative and qualitative measures were used; measures were cross-checked whenever possible; one group's contributions were confirmed by the other group.
- *Interaction of selection and treatment*: One representative of each expected role was allocated to each group and all participants answered a profile questionnaire to check whether they were really appropriate representatives of the roles they were expected to have.

A discussion of further threats (*reliability of treatment implementation*, *diffusion or imitation of treatments, hypothesis guessing, compensatory rivalry, and resentful demoralization*) can be found in [13]. As is common in (quasi-)experiments, some validity threats had to be accepted:

- *Low statistical power*: The number of subjects in this quasi-experiment made it impossible to perform any statistical analysis.
- *Random heterogeneity of subjects*: As the allocation of people to the treatments was based on convenience, the two groups might not have had similar knowledge and backgrounds.
- *Selection*: Participants were selected by the research program leader according to the expected profiles; the PLEvo-Scoping expert, who was not a Fraunhofer employee at that time, had had previous contact with those two participants; the PuLSE-Eco expert was from the same organization as the quasi-experiment participants.

We decided to deal with the lack of statistical tests as proposed by Yin [15], who claims that an effective way of strengthening the results of empirical studies when no statistical test is applicable is to perform them further. Table 1 and its related comments and interpretation (see Subsection 4.5, first paragraph) show that the two groups were comparable indeed, and people with the same role in the different groups had similar profiles (knowledge and background). Therefore, the threat of *Random heterogeneity of subjects* did not appear to be real. Concerning the threat of *Selection*, a question in the profile questionnaire was added that asked about the participant's motivation for taking part in the scoping process of the AAL platform. In addition, the scoping activities were conducted as they would have been conducted with any external customer. Ultimately, this threat did not appear to be real either, because some of the worst evaluations were made by one of the two quasi-experiment participants who had previous contact with the PLEvo-Scoping expert. Regarding the PuLSE-Eco expert, it should be noted that PuLSE-Eco was not the object of study of this quasi-experiment and any possible bias would have affected both treatments.

4.3 Operation

The quasi-experiment took place in the form of one two-day workshop for each treatment. The first part of each workshop was dedicated to the presentation of the application domain, the quasi-experiment's task, as well as relevant information for assuring a common understanding of the PL to be scoped. After that, each member of the scoping team completed the profile questionnaire.

Due to time restrictions, the PLEvo-Scoping expert suggested that the groups divide tasks in some activities according to the participants' role. Group 1 used this approach when identifying and characterizing facts (activity 4, part of the step *Environment Change Anticipation*), when identifying adaptation needs (activity 7, part of the step *Change Impact Analysis*), and when performing the step *PL Evolution Plan* as a whole. Group 2 decided to perform all activities as a group.

Group 2 received extra training and extra time to improve their lists of facts and adaptation needs because the initial number of these was very low (15 and 9, respectively). As scoping the AAL platform PL was a real problem, the goal of this quasi-experiment from the viewpoint of the leader of the AAL research program was to get the highest number and the best quality of results possible from each group. During the analysis of the impact of the adaptation needs on the set of PL features (part of activity 8, in the *Change Impact Analysis* step), the method expert asked Group 1 to define a criterion to distinguish unstable features from stables ones, based on the number of adaptation needs causing changes in the PL features. Group 1 defined that features affected by at least five adaptation needs would be considered unstable. The method expert asked Group 2 to adopt the same criterion.

Another remark concerning the quasi-experiment execution is related to the activities of analyzing the alternative solutions for dealing with relevant adaptation needs and selecting the best alternatives, which are part of the *PL Evolution Plan* step (see Section 2, activities 11 and 12). Due to time restrictions, only the most appropriate alternative solution for each adaptation need was analyzed.

4.4 Data Analysis

Table 1 summarizes the profiles of the quasi-experiment participants. In order to make data summarization and presentation easier, the 5-point ordinal scales were converted into a numerical scale (the higher the number, the better experience, knowledge, or motivation). The domain expert from Group 1 who had the technical viewpoint (column Group 1 - TE) did not answer the question about his capability of providing an overview of the AAL PL and its goals. This has little relevance for the data analysis, because it was not expected that the technical representative would have this capability.

Table 1. Profile of the scoping teams

Profile Item	**Group 1**			**Group 2**		
	TE	ME	PLM	TE	ME	PLM
Experience in AAL	3	4	4	2,5	3	4
Knowledge of the AAL platform	5	2	2	5	3	2
Experience in PL Scoping	2	2	1	1	2	4
Technical knowledge in the AAL context	4	3	4	3	4	4
Market knowledge in the AAL context	2	4	4	3	4	4
Capability of providing an overview of the AAL PL and its goals		4	3	2	3	4
Motivation	4	4	4	4	4	5

TE: domain expert with technical viewpoint; ME: domain expert with market viewpoint; PLM: product line manager.

Table 2 presents a quantitative overview of the scoping process results. The values in parentheses represent the number of facts or adaptation needs that had been given as examples during the extra training and were confirmed by Group 2.

Table 2. Quantitative overview of the workshops

PuLSE-Eco	**Group 1**	**Group 2**
Number of features	52	57
Number of products	7	7
Number of assessed domains	8	7
PLEvo-Scoping	**Group 1**	**Group 2**
Number of facts	30	15 + (11)
Number of adaptation needs	20	12 + (3)
Number of adaptation needs that were planned	8	5

Data Analysis related to Adequacy. Table 3 shows the participants' subjective evaluation of the three levels of support provided by PLEvo-Scoping (see legend at the bottom of Table 3 for details). Each cell gives the number of participants from a certain group who selected the value on the left side to evaluate the support in question. No participant selected the value "Not necessary, nor sufficient" for any level of support, which would be equivalent to 1 on the numeric scale. Participants were asked to indicate missing or annoying information when selecting those values. The annoying (not necessary, but required or provided) information related to Perception of Support 1

(column Support 1 - ST1) was "The relationship between actors' goals and facts is not so clear. Actors, facts, and adaptation needs would have provided enough support". The missing (necessary, but not required or provided) information related to Perception of Support 3 (column Support 3 - ST2) was "The basic alternative solutions have to be tailored to the concrete situation, which makes the activity difficult". Therefore, the values of *# Annoying Information 1* for Group 1 and *# Missing Information 3* for Group 2 were both 1, while the remaining quantitative metrics related to missing and annoying information had the value 0 (see conditions 2 and 3 in Table 5).

Table 3. PLEvo-Scoping support

Selected Value	Support 1		Support 2		Support 3	
	ST1	ST2	ST1	ST2	ST1	ST2
Missing information (2)						1
Annoying information (3)	1					
Mostly necessary and sufficient (4)	1	2	2	2	1	1
Necessary and sufficient (5)			1	1	2	1
Support 1: support for identifying relevant adaptation needs Support 2: support for deciding when to introduce an adaptation need and to which products Support 3: support for analyzing the alternative solutions.						

From Table 3, one can calculate, by converting the original ordinal values obtained from the questionnaires into numeric values and applying the arithmetic mean, the values of *Perception of Support 1* (3.5 for Group 1 and 4 for Group 2), *Perception of Support 2* (4.33 for both Group 1 and Group 2), and *Perception of Support 3* (4.67 for Group 1 and 3.67 for Group 2). All of them correspond to the ordinal value of either "Mostly necessary and sufficient" (4) or "Necessary and Sufficient" (5).

Table 4 shows the values obtained for the quantitative metrics. Adaptation needs and technical progress items were not confirmed by the other scoping team if considered irrelevant or outside the scope of the AAL platform PL. Technical progress items are facts related to technology evolution that the PL organization wants to pursue in order to get innovative features to the market as soon as possible.

Table 4. Values of quantitative metrics

Metrics	Group 1	Group 2
# Added Features	12	3
# Confirmed Added Features	7	2
# Technical Progress Items	8	12
# Confirmed Technical Progress Items	6	6
# Unstable Features	3	8
# Change in Priority due to Volatility	2	0

A few days after the workshop, the quasi-experiment participants were provided with a report on the respective results and asked to provide their perception of PLEvo-Scoping's adequacy using a 5-point ordinal scale. The ordinal values were converted into numeric ones, and the arithmetic mean was applied. The values obtained for the metric *Perception of Adequacy* were 4.5 (High) according to Group 1, and 4.33

(Medium to High) according to Group 2. No participant selected the values "Low" (1) and "Low to Medium" (2) for the adequacy of the method.

From the defined acceptance criterion (see Table 5) and the above data analysis, proposition P1 (*PLEvo-Scoping is adequate*) was accepted in the context of both groups. While Group 1's results even satisfied the non-obligatory condition (condition 5), the results of Group 2 did not satisfy it.

Table 5. Acceptance criterion for Adequacy

(1) *Perception of Support 1*, *Perception of Support 2* and *Perception of Support 3* ∈ Positive-Scale1, and
(2) (*# Missing Information 1* + *# Missing Information 2* + *# Missing Information 3*) <= 4 or they can be easily added (1 day of effort), and
(3) (*# Annoying Information 1* + *# Annoying Information 2* + *# Annoying Information 3*) <= 4 or they are already indicated as optional information, and
(4) *Perception of Adequacy* ∈ Positive-Scale2, and
(5) ideally, but not obligatory, *# Changes in Priority due to Volatility* / *# Unstable Features* > 0,50

where
- Positive-Scale1 = {Mostly Necessary and Sufficient, Necessary and Sufficient}, and
- Positive-Scale2 = {Medium to High, High}.

Data Analysis related to Feasibility. The effort for applying both approaches (PuLSE-Eco and PLEvo-Scoping) is described in Table 6, which neither takes into consideration the learning effort nor the scoping experts' effort. The learning effort related to PLEvo-Scoping was 2.48 person-hours for Group 1 and 2.68 person-hours for Group 2. The scoping experts' effort represents no loss of information for the quasi-experiment, because the PLEvo-Scoping expert herself did not carry out any activity. The values in parentheses in Table 6 refer to the extra time Group 2 used to improve their lists of facts and adaptation needs, after additional training by the PLEvo-Scoping expert. The PL manager's effort (column PLM) is also presented factored out of the total effort, because the PL manager from Group 2 managed to take part in more activities of the scoping process than the minimum previously established.

Table 6. Usage effort in person-hours

Phase	**Group 1**	**PLM**	**Group 2**	**PLM**
Product Line Mapping	5.95	1.98	8.60	2.87
Domain Potential Assessment	5.23	0.20	4.02	0.35
Total PuLSE-Eco	11.18	2.18	12.62	3.22
Preparation for Volatility Analysis	0.93		1.50	0.50
Environment Change Anticipation	2.70		3.83 + (0.32)	0.60
Change Impact Analysis	5.25	0.6	6.02 + (0.50)	0.63 + (0.2)
PL Evolution Plan	2.10	1.10	3.90	1.30
Total PLEvo-Scoping	10.98	1.7	15.25 + (0.82)	3.03 + (0.2)
Total Effort (PuLSE + PLEvo)	22.16	3.88	27.87 + (0.82)	6.25 + (0.2)

The quasi-experiment participants evaluated the difficulty of carrying out the PLEvo-Scoping activities. Again, conversion of ordinal values into numeric ones and the arithmetic mean were used. The values obtained for *General Perception of*

Difficulty were 3.27 for Group 1 and 2.83 for Group 2, which represent the ordinal value "Neither Difficult nor Easy".

Furthermore, the quasi-experiment participants were asked to give their perception of the feasibility of PLEvo-Scoping using a 5-point ordinal scale, after being provided with the effort metrics and the method results. The ordinal values were converted into numeric ones, and the arithmetic mean was applied. The values obtained for the metric *Perception of Feasibility* were 4 (Medium to High) according to Group 1, and 3.67 (Medium to High) according to Group 2. No participant selected the values "Low" (1) and "Low to Medium" (2) for the method's feasibility.

Taking into consideration the acceptance criteria defined in Table 7 and the reported data analysis, proposition P2 (*PLEvo-Scoping is feasible*) was accepted in the scope of both scoping teams.

Table 7. Acceptance criterion for Feasibility

(6) *General Perception of Difficulty* ∈ NDifficult-Scale, and
(7) *Perception of Feasibility* ∈ Positive-Scale2 where
– NDifficult-Scale = {Neither Difficult nor Easy, Easy, Very Easy}.

Feedback from the Quasi-Experiment Participants. The quasi-experiment participants were encouraged to register in the questionnaires any comments related to the support provided by PLEvo-Scoping or to the conditions in which the quasi-experiment was performed. The feedback collected was:

- They would like to have 1) more time to perform some activities, 2) a more detailed explanation of PLEvo-Scoping concepts and their relationships ("one slide was not enough"), 3) better tool support for applying the method. PLEvo-Scoping application in this quasi-experiment was supported by an Excel file and the quasi-experiment participants reported it was difficult to switch between different sheets during the execution of an activity.
- They would like to have "a list of adaptation needs that usually apply to all kinds of products or development" (see Section 2, activity 7).
- They reported the difficulty of estimating the business impact and the technical risk of an adaptation need and suggested registering the rationale behind the assignment of values to the attributes (see Section 2, activity 8).
- When elaborating the *PL Evolution Map* (see Section 2, activity 10), it would be useful to have the relationships between the adaptation needs.
- They think "further possibilities to deal with adaptation needs would be reasonable"; alternative solutions are probably missing (see Section 2, activity 11).

Additionally, after having applied PuLSE-Eco only, Group 1 said that it was not easy to think about technical progress, because "it is quite different from thinking about domains, products, and features. We need to switch our minds". PLEvo-Scoping is expected to help them do that.

4.5 Comments and Interpretation of Results

From the values presented in Table 1, we concluded that each participant had the competencies required to perform the role he/she had been assigned to by the AAL

research program leader (technical knowledge on the part of the domain expert with the technical viewpoint, market knowledge on the part of the domain expert with the market viewpoint, and the capability of providing an overview of the AAL PL on the part of the PL managers). The values related to *Motivation* were very similar. While Group 1 had higher values for *Experience in AAL*, Group 2 had higher values for *Knowledge of the AAL platform*. Therefore, overall, their domain knowledge can be considered similar as well. The main difference between the two groups is related to *Experience in PL Scoping*, because the PL manager of Group 2 had already participated in a scoping process. As the time spent by the PL managers in the workshops was limited, as PLEvo-Scoping was a new method, and as Group 2 applied the interwoven approach, we do not believe this experience had much influence. Consequently, we considered the two groups to be comparable.

The number of facts and adaptation needs identified by Group 2 (see Subsection 4.3 and Table 2), which initially was too low, and the subsequent extra training and complementary activities to improve it may have been caused by Group 2's choice of performing all activities as a group, not allowing any parallelism. Another factor that may have influenced Group 2's performance is the interweaving of activities from both methods, because the group had to switch their minds between scoping and evolution activities. Despite the differences in Tables 2 and 4, which are justified above, the two approaches for integrating the method into an existing PL scoping process showed similar general results: The method could be applied in just one day, and it was considered adequate, feasible, and neither difficult nor easy to apply.

Concerning the annoying information that was pointed out by a member of Group 1 (see Subsection 4.4 - Data Analysis related to Adequacy), the identification of actors' goals is optional and therefore cannot be considered an annoying request. Furthermore, this information was very useful when Group 2 had identified only 15 facts and 9 adaptation needs and needed some help. The method expert used the actors' goals that had been identified by the group to derive some possible examples of facts and adaptation needs; some of these were considered relevant by the group and accepted. The missing information that was pointed out by a member of Group 2 is really missing information and cannot be added easily. The best way to address it is to build an experience base of alternative solutions for the PL organization.

Contrary to our expectations, the interweaving of activities in treatment 2 did not provide any benefit. We expected that some activities of PuLSE-Eco (especially, the activity *Assess Domains*) would provide insights into some PLEvo-Scoping activities and vice versa, but the workshop format (two consecutive days) and time pressure did not allow the quasi-experiment participants to really benefit from previous activities. The interaction between the two methods must be investigated further.

With regard to the difficulty of applying PLEvo-Scoping, the value for *General Perception of Difficulty* (Neither Difficult nor Easy) is acceptable due to the inherent difficulty of some activities. Furthermore, this result might have been influenced negatively by the short time available for training.

In order to address the feedback obtained from the quasi-experiment participants, we have included in the PLEvo-Scoping activities of characterizing adaptation needs (Section 2, activity 8) and analyzing alternative solutions (Section 2, activity 11) the register of the rationales behind the assignment of values to the attributes. We plan to provide tool support for applying PLEvo-Scoping, which should make the registered and

derived relationships between adaptation needs more explicit for the scoping team when elaborating the *PL Evolution Map* (Section 2, activity 10). Concerning the problem of missing alternative solutions, we want to continue to investigate alternative solutions for dealing with adaptation needs together with PL architects and, at the same time, start to collect real occurrences in an experience base of alternative solutions. However, we do not believe it is possible to compile "a list of adaptation needs that usually apply to all kinds of products or development", because adaptation needs are expected to be application-specific and change over time. PLEvo-Scoping provides a list of 35 generic facts instead, so that the scoping team can reflect on whether and how they may apply to the application domain and those facts should lead to the adaptation needs. Moreover, the difficulty of estimating the business impact and the technical risk of an adaptation need cannot be addressed completely, because it is inherent to the problem. Depending on the experience the scoping team has in the domain and with the required technologies, this difficulty is higher or lower.

5 Conclusion

PLEvo-Scoping is a method for supporting PL scoping teams in systematically reasoning about the driving forces of evolution in a certain domain, especially reasoning about who is behind these forces and how their decisions, needs, or achievements may affect the PL infrastructure. Our method allows the PL scoping team to proactively identify and prioritize the adaptation needs that will probably be required in the PL infrastructure and decide about how to deal with them.

The contribution of this paper is the characterization of the adequacy and feasibility of PLEvo-Scoping in practice, meaning according to professionals in charge of scoping an AAL PL. A quasi-experiment was performed to obtain feedback from PL practitioners on how to improve the method and to provide first empirical data on the usage of PLEvo-Scoping, so that other PL organizations can decide on whether to try out the method or not. The method could be applied in just one day, and overall, the quasi-experiment participants perceived it as being adequate and feasible. Those results were really positive, taking into consideration that predicting the future is hard, the quasi-experiment participants applied the method for the first time, the learning effort had to be minimal, and there was no specific tool support.

Although PLEvo-Scoping was applied for just one day, we recommend two to three days for its application, in order to give the PL scoping team enough time to understand the method's underlying concepts and carry out its activities without so much time pressure. This recommendation addresses two of the comments provided by the quasi-experiment participants. In addition, PLEvo-Scoping can also be applied interactively, where the method expert would guide the PL scoping team in performing their activities. The intervention of the PLEvo Scoping expert in this quasi-experiment was kept to a minimum in order not to affect the results.

We think that performing a quasi-experiment is a good means for providing empirical evidence on Product Line engineering technologies, because, compared to case studies and experiment, its intermediate degree of control makes it easier to have PL professionals as subjects while still allowing manipulation of variables and comparison of treatments on some level. In this way, it is possible to convince practitioners as

to the applicability of a method in real settings and, at the same time, to provide researchers with scientific evidence on the value of the method.

We intend to perform further empirical studies in order not only to corroborate the results reported in this paper, but also to further analyze the interaction between PLEvo-Scoping and the scoping approach.

Acknowledgments. The authors acknowledge the Alexander von Humboldt Foundation for its financial support and the BelAmi project (BMBF grant number HUN 04/A02) for the opportunity to evaluate PLEvo-Scoping.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2001)
2. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, Heidelberg (2005)
3. Knauber, P., Succi, G.: Perspectives on Software Product Lines. Software Engineering Notes 27(2), 40–45 (2002)
4. Savolainen, J., Kuusela, J.: Volatility Analysis Framework for Product Lines. In: Proc. SSR 2001, Toronto, pp. 133–141 (2001)
5. Bayer, J., Flege, O., Knauber, P., et al.: PuLSE: A Methodology to Develop Software Product Lines. In: Proc. SSR 1999, Los Angeles, pp. 122–131 (1999)
6. Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Analyzing Software Architectures for Modifiability. TR HK-R-RES00/11-SE, University of Karlskrona/Ronneby. Ronneby (2000)
7. Schmid, K.: Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines. PhD Theses in Experimental Software Engineering. Fraunhofer IRB (2003)
8. John, I., Knodel, J., Lehner, T., et al.: A Practical Guide to Product Line Scoping. In: Proc. SPLC 2006, Baltimore, pp. 3–12 (2006)
9. Villela, K., Dörr, J., Gross, A.: Proactively Managing the Evolution of Embedded System Requirements. In: Proc. RE 2008, Barcelona, pp. 13–22 (2008)
10. John, I., Villela, K., Gross, A.: AAL Platform Product Line – Scoping Results and Recommendations. TR 074.09/E, Fraunhofer IESE, Kaiserslautern (2009) (available upon request)
11. Zelkowitz, M., Wallace, D.: Experimental Models for Validating Technology. IEEE Computer 31(5), 23–31 (1998)
12. Basili, V., Caldiera, G., Rombach, H.: Goal Question Metrics Paradigm. Encyclopedia of Software Engineeering 1, 528–532 (1994)
13. Villela, K., John, I.: Usage of PLEvo-Scoping in the Ambient Assisted Living Domain: A Quasi-Experiment Package. TR 093.09/E, Fraunhofer IESE, Kaiserslautern (2010)
14. Wohlin, C., Runeson, P., Höst, M., et al.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell (2000)
15. Yin, R.: Case Study Research: Design and Methods, 3rd edn. Sage Publications, Thousand Oaks (2003)

Challenges in Aligning Requirements Engineering and Verification in a Large-Scale Industrial Context

Giedre Sabaliauskaite[1], Annabella Loconsole[1], Emelie Engström[1], Michael Unterkalmsteiner[2], Björn Regnell[1], Per Runeson[1], Tony Gorschek[2], and Robert Feldt[2]

[1] Department of Computer Science, Lund University, Sweden
[2] Blekinge Institute of Technology, Sweden
{Giedre.Sabaliauskaite,Annabella.Loconsole,Emelie.Engström,
Björn.Regnell,Per.Runeson}@cs.lth.se
{Michael.Unterkalmsteiner,Tony.Gorschek,Robert.Feldt}@bth.se

Abstract. [**Context and motivation**] When developing software, coordination between different organizational units is essential in order to develop a good quality product, on time and within budget. Particularly, the synchronization between requirements and verification processes is crucial in order to assure that the developed software product satisfies customer requirements. [**Question/problem**] Our research question is: what are the current challenges in aligning the requirements and verification processes? [**Principal ideas/results**] We conducted an interview study at a large software development company. This paper presents preliminary findings of these interviews that identify key challenges in aligning requirements and verification processes. [**Contribution**] The result of this study includes a range of challenges faced by the studied organization grouped into the categories: organization and processes, people, tools, requirements process, testing process, change management, traceability, and measurement. The findings of this study can be used by practitioners as a basis for investigating alignment in their organizations, and by scientists in developing approaches for more efficient and effective management of the alignment between requirements and verification.

Keywords: requirements engineering, software verification, software testing, coordination.

1 Introduction

Are we sure that the tests performed are based on requirements and not on technical specifications supplied by developers? Are we sure that the test coverage is adequate? In order to assure that customer requirements are realized as intended these questions must be asked and answered. However, this is not an easy task, since requirements tend to change over time [13], and in many cases the requirement specifications are not updated during the development of a product making it hard to use them as a solid base for creating e.g. test cases [7, 15]. In small systems with just a few requirements it could still be possible to handle the changes manually, but it gets extremely hard in

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 128–142, 2010.

complex systems with thousands of requirements. Therefore, there is a need for a mechanism to manage coordination between the requirements and the verification processes. We call such coordination *alignment*.

In this paper, we examine the challenges in aligning the requirements and the verification processes. We present preliminary results of an interview study performed in a large software company in Sweden. The overall goal of our research is to understand how alignment activities are performed in practice, what the important problems are and what can be improved to gain better alignment. The results presented in this paper are a set of challenges that can help practitioners and researchers. Practitioners can, for instance, allocate more resources in the areas that are challenging when aligning the requirements and the verification processes. Researchers can also benefit from our results by focusing their research on the areas that are the most challenging. The results are valid in the context of the company under investigation. We are currently extending the case study to other companies. By comparing the results of this case with other case studies it will be possible to get a more general picture of challenges in different kinds of organizations and in different domains.

The paper is structured as follows. In Section 2, we present related work in the area. In Section 3, we describe the research approach used in this qualitative case study. Section 4 describes the alignment challenges found. Finally conclusions and future work are presented in Section 5.

2 Related Work

In [17], authors presented the findings of the discussions with test managers and engineers in software development organizations regarding difficulties of integrating independent test agencies into software development practices. The company where we have performed interviews does not commonly use independent test agencies, however it has separate requirements, development, and testing units. Therefore it would be interesting to compare the results of having independent test agency and independent test unit within the company under study.

Findings related to change management emphasize the importance of synchronization between the development and test with respect to modifications of functionality [17]. The results of our study confirm these findings. One of the most recurrent challenges identified in our study is that requirements are not being updated on time.

Findings related to people interactions and communication stress the need of communication between development and test organizations. If testers do not know who wrote or modified the code, they do not know whom to talk to when potential faults are detected. On the other hand, it could be difficult for developers to inform testers on upcoming functionality changes, if they don't know whose test cases will be affected [17]. Our study confirms these results as well. Most of the interviewees suggest that alignment could be greatly improved if requirements and testing people would interact more with each other.

Several surveys on requirements related challenges are present in the literature: problems in the requirements engineering process [9], requirements modeling [6], quality requirements [7], requirements prioritization [1], and requirements interdependencies [8]. Among these, Karlsson et al. [1] have results similar to ours, i.e. tool integration is difficult and it is a challenge to write quality requirements.

Most of the studies above do not focus on the alignment between the requirements and the verification processes. Research in connecting requirements and testing has been performed by several authors, for instance Uusitalo et al. [4], Post et al. [3], and Damian and Chisan [10]. Uusitalo et al [4], have conducted a series of interviews in order to investigate best practices in linking requirements and testing. Among the best practices, authors mention early tester involvement in requirements activities. They conclude by suggesting to strengthening the links between requirements engineers and testers, since it is difficult to implement traceability between them; a conclusion supported by this study (see Section 4.7).

The importance of linking requirements and verification is also stressed by Post et al. [3]. They describe a case study showing that formalizing requirements in scenarios make it easier to trace them to test sets. Damian and Chisan [10] present a case study where they introduce a new requirements engineering process in a software company. Among the practices in the process, they include traceability links between requirements and testing, cross-functional teams, and testing according to requirements. They show that an effective requirements engineering process has positive influence on several other processes including testing process.

The case studies above [3, 4, 10] are performed in a medium scale requirements engineering context [11], while our study is performed in a large/very large scale context and includes many aspects of aligning requirements and verification.

3 Research Approach

The approach used in this study is qualitative. Qualitative research consists of an application of various methods of collecting information, mainly interviews and focus groups. This type of research is exploratory [16]. Participants are asked to respond to general questions, and the interviewers explore their responses to identify and define peoples' perceptions and opinions about the topic being discussed. As the study was meant to be deep and exploratory, interviews were the best tool since surveys are not exploratory in nature. The interviews were semi-structured to allow in-depth, exploratory freedom to investigate non-premeditated aspects.

In this study, we interviewed 11 professionals in a large software development company in Sweden, based on the research question: What are the current challenges in aligning the requirements and the verification processes?

The viewpoint taken in this research is from a process perspective. The researchers involved do not work directly with artifacts, but with processes and have expertise in fields like requirements, testing, quality, and measurement.

Based on our pre-understanding of the processes involved in aligning requirements and verification, a conceptual model has been designed (see Figure 1). This model was used as a guide during the interviews. In this model, we consider three dimensions of requirements and test artifacts, connected through work processes. One is the *Abstraction level dimension*, from general goals down to source code, which is similar both for the requirements and the testing side. Test artifacts are used to verify the code, but also for verifying the requirements. The arrows are relationships that can be both explicit and implicit, and can be both bi- or uni-directional. Then, we have the *Time dimension*, in which the processes, the products, and the projects change and

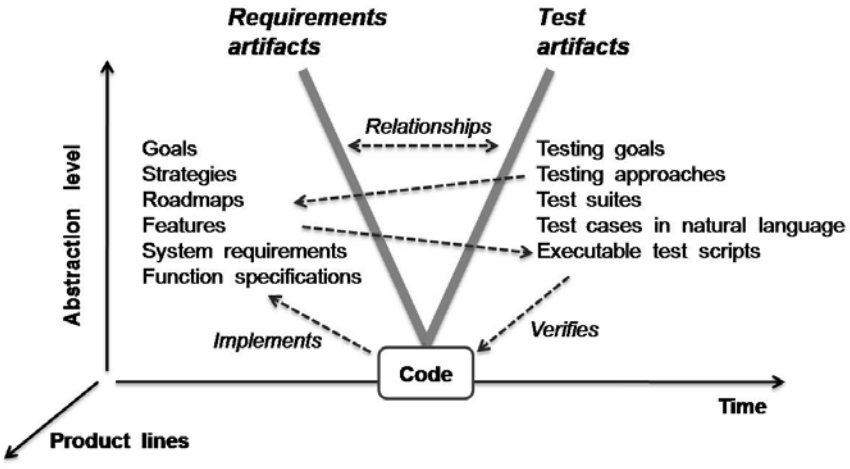


Fig. 1. Conceptual Model

evolve. This has an effect on the artifacts. There is also the *dimension of Product lines*, which addresses variability, especially applicable when the development is based on a product line engineering approach [2].

Case Context. Our results are based on empirical data collected through interviews at a large anonymous company, which is using a product-line approach. The company is developing embedded systems for a global market, and has more than 5000 employees. A typical project in this company lasts about 2-years, involves circa 800-1000 men per year, and has around 14000 requirements and 200000 test cases. The tool DOORS is used for requirements management, and the tool Quality Center for test management. Further information about the company is not disclosed for confidentiality reasons.

The interviews have been distributed in time between May and October 2009.

3.1 Research Methodology

In this study, challenges and problems, as well as, current good practices and improvement suggestions regarding alignment between the requirements and verification processes have been identified through interviews with software engineering practitioners. The results from 11 interviews are included in this paper. Employees with different roles have been interviewed: quality management related roles (quality manager and quality control leader), requirements related roles (requirements process manager, requirements architect and requirements coordinator), developer and testing related roles (test leader, tester). The research was conducted in several steps:

1. Definition of interview guide;
2. Interview planning and execution;

3. Transcription of interviews, and division of transcriptions into sections;
4. Definition of codes (keywords) to be assigned to transcriptions' sections;
5. Coding of interview transcriptions using predefined codes;
6. Sorting of coded transcriptions to group transcription sections according to codes;
7. Analysis of the results;
8. Validation of results by feedback to the organization.

Step 1. We constructed an interview guide, which is a document containing 30 questions to be asked during the interviews. The first version of the interview guide contained 22 questions, defined based on the research questions. The questions were validated during 2 pilot interviews. This led to the updated list of questions, grouped into several topics, such as general questions about requirements and testing, questions on quality requirements, etc. An overview of the interview guide is available in Table 1[1].

Table 1. Overview of the interview guide

Interview topics	Description
Software requirements	Handling of functional and quality requirements, customer involvement
Software testing	Handling of testing artifacts, customer involvement, testing of functional and quality requirements
Alignment between requirements & verification processes	Alignment importance, current alignment method (documents, processes, methods, tools, principles, practices, etc.), alignment responsible, problems & challenges, improvement ideas & expected benefits
Measurements and feedback gathering	Alignment related measurements, performance indicators, customer satisfaction evaluation
Product line engineering	Handling of requirements and testing, maintaining alignment
Outsourcing	Maintaining alignment in case of outsourcing

Step 2. Eleven professionals were interviewed; each interview lasted for about one and a half hour. All interviews were recorded in audio format and notes were taken. A semi-structured interview strategy [16] has been used in all interviews, where the interview guide acted as a checklist to make sure that all important topics were covered. 2-3 interviewers interviewed one interviewee. One of the interviewers lead the interview, while the others followed the interview guide, took notes, and asked additional questions. The selection of the interviewees has been made based on recommendations by requirements managers, test managers, and the interviewees themselves. (At the end of each interview we asked the interviewees if they could recommend a person or a role in a company whom we could interview in order to get alignment related information).

Step 3. Interviews were transcribed into text in order to facilitate the analysis. The transcriptions were then divided into text sections containing 1-2 sentences. All the

[1] The complete version of the interview guide and coding guide are available at: http://serg.cs.lth.se/research/experiment_packages/interview_study_on_requirements_verification_alignment/

text sections have been numbered in order to keep the order of the sentences. The size of the transcriptions ranged from 4000 words to about 9000 words per interview.

Step 4. As suggested by C.B. Seaman [12], codes (keywords) were assigned to the transcriptions' sections in order to be able to extract all the sections related to a specific topic. However, the definition of the coding scheme turned out to be a non-trivial task. We started by making an initial list of possible codes, which included codes related to our research questions, alignment methods, quality requirements [14] and software development process activities. In order to extend and tailor this initial list of codes to our interview context, we decided to perform exploratory coding [16], which included six researchers analyzing several interview transcriptions individually and assigning suitable codes to the text sections.

The result of exploratory coding was a list with 169 codes. In the next stage, we reviewed the codes resulting from the exploratory coding, grouped them into several categories at different abstraction levels and developed a coding guide. The coding guide is a document containing the list of codes and detailed instructions of how to code a transcription. In order to validate the coding guide, seven researchers used it to code the same interview transcription (let's call it X) individually, and then had a meeting to discuss differences in coding and possible improvements of the coding guide. Kappa inter-rater agreement [18] has been used as a metric to evaluate improvement in homogeneity of coding by different researchers. Consequently, the coding guide was updated and the interview transcription (X) was coded again using the updated version of the coding guide to make sure that the differences between different coders were minimized. The coding guide included codes at three abstraction levels: high, medium, and low (see Table 2). The high-level codes were based on research questions. The medium-level codes included different categories relevant to our research, and the low-level codes were the coder's interpretation of the transcription's section. A summary of the codes is presented in Table 2.

Table 2. Overview of the codes assigned to transcription's sections (see footnote 1 for a complete list of codes)

Abstraction level	Description
High	Codes related to research questions, i.e. alignment practices, problems and challenges, improvement ideas and benefits
Medium	Two groups of codes: Group 1 – thirteen categories, which include requirements, testing, traceability, configuration management, organization processes, interactions, product quality aspects, and measurements among others. Group 2 – additional categories, e.g. product-line engineering, outsourcing, open source.
Low	Coder's interpretation of the transcription's section, a brief summary of the information described in the section

Step 5. Eleven interview transcriptions were randomly assigned to four researchers, who coded them using the final version of the coding guide. The template used during coding is shown in Table 3.

Table 3. Template used during coding

No	Text	High-Level Coding	Medium-Level Coding			Low-Level Coding, Comments
		Research Questions	Group 1		Group 2	
			Primary	Secondary		

Step 6. Coded interview transcriptions were merged into one file, making it possible to group transcription sections according to codes.

Step 7. The identified transcription's sections of each group were analyzed by two researchers. In order to identify alignment challenges, researchers studied all the transcription's section coded as "challenges" with the goal to extract challenges from the information provided by interviewees. Some challenges were similar and therefore could be reformulated or merged together, while others were kept apart as they were different.

Step 8. The results of the analysis were validated by feedback from the organization where the interviews have been conducted.

3.2 Validity Discussion

A discussion of possible threats to validity will help us to qualify the results and highlight some of the issues associated with our study. As suggested by P. Runeson and M. Höst [5], we have analyzed the construct validity, external validity, and reliability. Internal validity is concerned with threats to conclusions about cause and effect relationships, which is not an objective of this study. A detailed list of possible threats is presented in [16].

Threats to Construct Validity. The construct validity is the degree to which the variables are accurately measured by the measurement instruments used in the study [5]. The main construct validity threat in this study regards the design of the measurement instrument: are the questions formulated so that the interviews answer our research questions? Our main measurement instrument is the interview guide (see Section 3.1, Step 1), which includes the questions to be asked. Two researchers have constructed it by analysing the research questions and creating sub-questions. Five researchers have reviewed it to check for completeness and consistency; therefore we believe that the interview guide is accurate. The other measurement instrument is the coding guide. As described in Section 3.1, Step 4, this instrument has been validated by seven researchers in order to make sure that the result of the coding activity had minimal individual variations.

The questions in the interview guide were tailored on the fly to the interviewees since the professionals participating in the interviews had different roles and different background. Our study is qualitative; the goal is not to quantify answers of the same type, rather to explore the different activities in the company, which could be done best by investigating deeply the role of each interviewee.

Another potential threat in this study is that different interviewees may interpret the term "alignment" differently. For this reason, the conceptual model (see Figure 1) has been shown to the subjects during the interviews, in order to present our definition of alignment between requirements and verification.

Threats to External Validity. The threats to external validity concern generalisation. The purpose of this study is not to do any statistical generalization of the results to other contexts, but to explore the problems and benefits of alignment in the context of the specific company. The study was performed in an industrial environment where the processes were real, and the subjects were professionals. Hence, we believe that the results can be analytically generalized to any company of similar size and application domain. The company might not be representative; therefore more companies will be interviewed in order to get results independent of the kind of company.

Reliability. Reliability issues concern to what extent the data and the analysis are dependent on the researchers. Hypothetically, if another researcher later on conducts the same study the results should be the same. In this study, all finding have been derived by at least two researchers, and then reviewed by at least three other researchers. Therefore, this threat has been made smaller.

In our study, the investigation procedures are systematic and well documented (see Section 3). The interview guide, the researchers' view (the conceptual model), and the coding scheme were reviewed independently by seven researchers with different background.

The presented observations reflect the views of the participants. The interviews have been recorded and transcribed. The transcriptions could contain errors due to misinterpretation, mishearing, inaccurate punctuation or mistyped words. In order to minimize these threats, the transcriber has also been present at the interview. Moreover, the transcriptions were sent to the interviewees so that they could correct possible misinterpretation of their answers.

One factor affecting the reliability of the data collected can be the fact that the interviews capture the subjective opinion of each interviewee. However, we interviewed 11 professionals, which we believe is a sufficient amount to capture the general view of the company. Influence among the subjects could not be controlled and we could only trust the answers received. The choice of the subjects in the company might not give a representative picture of the company; however, the subjects had different roles and we tried to cover diverse roles within the company.

Regarding the coding activity, it is a classification of pieces of text, which are taken out of context; hence there is a risk of misinterpretation. This risk was minimized by checking the whole context of the text while doing data analysis.

To summarize, we believe that the validity threats of our results are under control, although the results should not be generalized to all organizations.

4 Analysis and Result

The result of this study includes a range of challenges faced by the studied company grouped into these categories: organization and processes, people, tools, requirements

process, testing process, change management, traceability, and measurement. The grouping is rough: if the challenge belonged to several categories, we assigned it to the category which was the most relevant. The choice of the categories was based on the medium level codes (see Table 2). All challenges are rooted in the interview transcriptions. The challenges of each group are presented in Subsections 4.1-4.8.

4.1 Organization and Processes Related Issues

This section summarizes the alignment problems and challenges related to the company's organizational structure and processes.

- The requirements and verification processes are separate processes and are not aligned. Furthermore, processes can use different standards of documentation, which negatively influence the hand-over between different parts of organization. Moreover, some parts of the company follow a documented development process while other parts do not.
- Frequent process changes negatively influence alignment. It would take time for people to learn and use the new process. Sometimes, people are reluctant to use a process knowing that it will change soon. Also, some good practices could be lost due to the process changes.
- Distance in time between the development of requirements and test artifacts can create alignment problems. Requirements can be approved without having test cases associated with them. This can result in having non-testable requirements.
- In a large company, gaps in communication across different organizational units often occur, especially at the high level. Furthermore, as stated by an employee "*it is hard to find who is accountable for things because depending on who you ask you get very different answers*". Therefore, this could affect the alignment, especially at the high abstraction level of the requirements and verification processes.
- Implementation of process improvements is time consuming, especially when the improvements are involving several units. Several issues related to the management can affect the alignment, e.g. decisions are not documented, lessons learnt are not always collected and processes depends on individual commitment.

Summarizing the challenges, the requirements and the verification processes are not aligned and are distant in time. There are also communication problems across different organizational units and the decisions are not documented, therefore it is hard to know who is accountable for a decision. The organizational structure and the processes, as well as changes in these are influencing the alignment. One reason could be that the company is very large and many organizational units are involved, and not every unit follows the documented process, and the standard for documentation.

4.2 People Related Issues

This subsection presents a list of issues that are related to people, their skills and communication with each other.

- Professionals do not always have good technical knowledge and understanding about the work of other units. Requirements engineers sometimes lack knowledge

about implementation as well as testing, while testers lack knowledge of requirements. Also, professionals are sometimes unwilling to move within the company in order to gain this knowledge. This has a negative effect on alignment between requirements and verification processes.

- Lack of cooperation between requirements people, developers and testers is affecting the alignment. In some cases, requirements engineers and developers have a good communication, as well as developers and testers. However, when there is a lack of direct communication between requirements and testing people, alignment is influenced negatively.

The main challenge in this area is communication, cooperation, and understanding of each other's work within the company. This can be hard when working under tight deadlines; there is no time to communicate and understand each other's work. Adequate technical knowledge, communication and cooperation between requirements people, developers and testers greatly influence the alignment.

4.3 Tools Issues

Software tools play a crucial role in maintaining alignment between different artifacts. The following are several tool related issues.

- The lack of appropriate tools influences the alignment. It is very important to have reliable and easy to use requirements and verification tools. If the tool is difficult to use, or it is not reliable, people are not willing to use them. Having a good requirements management tool, which includes not only information about requirements, but also the flow of requirements, is crucial for testers. Otherwise, testers try to get this kind of information from other sources, for instance the developers. Tools for managing quality requirements are needed, otherwise there is a risk that quality requirements are not implemented and/or tested.
- It is important to keep the requirements database updated. If requirements are not up to date, testers will test according to old requirements and will find issues, which are not really failures, but valid features.
- If there is no tool to collect customer needs, it is difficult to keep them aligned with requirements, hence with test cases as well. And this leads to misalignment between customer needs and requirements, and consequently affects customer satisfaction with the final product.
- In cases when requirements and testing artifacts are stored in different tools, there is a need of good interfaces between these tools, and access of all interested parties to the tools. Otherwise, it becomes very difficult to maintain alignment. Especially when there are many-to-many relationships between requirements and test cases.
- If the mapping between requirements and test cases is not presented in a clear way, it could contain too much redundant information, and therefore it could be difficult for requirements people and testers to use it.

Most of the interviewees stated the lack of adequate software tools, which would allow to handle requirements, verification, and to measure the alignment between them. Furthermore, the interface of the tools and tool integration is not always good.

The consequence of this is that people become reluctant to use them and do not update the information stored in them. This is greatly affecting the alignment.

4.4 Requirements Process Related Issues

This subsection presents a list of issues that are related to the requirements process.

- Requirements sometimes are not given enough attention and consideration by other organizational units, such as development and testing units. According to an employee "*Developers do not always review the requirements, and discover requirements that can not be implemented during development, even when having agreed on the requirements beforehand*". This could be due to the lack of involvement of developers and testers in requirements reviews.
- Not having a good way of managing customers' needs makes it more difficult to define requirements, especially requirements at a high abstraction level.
- Requirements engineers do not think about testability of requirements. Therefore, requirements could turn out to be non-testable.
- Dealing with quality requirements is a difficult task. Quality requirements tend to be badly structured or vague. Furthermore, it is difficult to assign quality requirements to different development groups for implementation, since several groups are usually involved in implementing a quality requirement, and none wants to take a full responsibility for that.
- It is difficult to maintain alignment in organizations working with a large set of requirements, when the number of requirements reaches tens of thousands or more. Furthermore, in the organizations, which are using a product lines engineering [2] approach, maintaining alignment between domain and application requirements and test cases could be a challenge.

As we can see, there are numerous challenges related to requirements process, which affect alignment. Most of the interviewees stress the importance of updating requirements as soon as changes occur, and finding adequate ways of defining and managing quality requirements. These two are the most recurrent requirements process related challenges.

4.5 Testing Process Related Issues

The following are the issues that related to the testing process.

- Sometimes testers lack clear directions on how to proceed with testing. Especially while testing high-level requirements, such as roadmaps for example. It is difficult to test that the products adhere to roadmaps, since such testing takes a long time and is costly. Usually short loops are preferred.
- In case several organizational units are involved in testing, the cooperation between them is crucial. It is particularly relevant to the companies, which have a product line engineering approach, since different organizational units could be performing domain and application testing, and the faults detected in applications should be removed from domain as well.

- There is a lack of verification at early development stages, especially of quality requirements verification. This results in lower quality of the product, as well as added cost and time spent on removal of defects at later development phases.
- It is inefficient to maintain alignment of requirements and test cases due to the large amount of test cases; sometimes their number reaches hundreds of thousands.
- It is difficult to get requirements people interested in having good quality test cases. Requirement people's involvement in reviewing test cases contributes to alignment, since this would help to assure that test cases comply with requirements.

As we can see from the above-mentioned challenges, there are numerous testing process related issues that can affect alignment. Having a well defined testing process at different development stages, and good cooperation between testing units could help improve the alignment.

4.6 Change Management Issues

The following are the challenges related to change management.

- It is sometimes difficult to find the people responsible, if a change occurs, if a defect is found, or if there is a need of further information. Thus, requirements engineers do not always inform related developers and testers in case of a requirements change. Furthermore, if a failure is found during maintenance phase, it is extremely difficult for maintenance people to find requirements people who can give information regarding requirements, or whom to inform about implemented changes. Therefore, maintenance people sometimes need to use testers as a source of information about requirements.
- There is a lack of strategy in deciding which changes to implement in case there is not enough time or resources to implement all changes.
- The information about changes is not always timely updated in the requirements database. Therefore it is difficult for developers and testers to know that the change has occurred.

Updating the requirements on time is one of the most recurrent challenges. It is therefore important to find ways to cope with changes immediately so that the traceability with testing can be maintained. In addition, delta handling and good tracking and reporting on the requirements and test case tools is needed to easily track changes and verify completeness.

4.7 Traceability Issues

The following are the challenges related to traceability between requirements and testing artifacts.

- There is a lack of links between requirements and test cases. Some test cases are very complex; therefore it is difficult to trace them back to requirements.
- If traceability between requirements and test cases is not maintained, testers keep testing requirements that have been removed. The reasons for lack of traceability

could be the difficulty to implement traceability, and the lack of resources to maintain it.
- Having large legacies implies that a lot of test cases do not have requirements linked. This complicates implementation of alignment.
- Ideally, alignment should be implemented and maintained at all abstraction levels of requirements and verification processes. However, if it cannot be done for various reasons, such as lack of resources or time constraints, it is necessary to clearly define at which level to implement alignment.

The main challenge is the large volumes and complexity of requirements, test cases and test results. These are negatively influencing traceability. Better tools could help in managing the traceability in large scale requirements engineering and testing.

4.8 Measurements Issues

The following are the measurements related challenges.

- Due to the lack of experience in using measurements, it is difficult to define appropriate metrics or indicators.
- There is a lack of alignment related metrics. For example, one of the alignment metrics is requirements coverage by test cases, which is measured by calculating a percentage of requirements that have associated test cases. However, if a requirement has several test cases associated to it, it still could lack complete test coverage. Therefore, additional metrics are needed in order to get more complete information about requirements coverage.
- Key Performance Indicators (KPI) and metrics should be appropriate at both operative, as well as, top management level. Sometimes KPIs are useful only at top management level, but do not provide important information at the operative level regarding the things that could be improved.
- Sometimes target values for metrics and indicators are defined without a business case, not based on historical measurement data. Therefore, they could be inachievable.

Among challenges regarding measuring the alignment that are mentioned, the most recurrent is the difficulty of defining metrics to measure the alignment, especially the requirements coverage. Definition and use of adequate alignment metrics could help improve the alignment.

5 Conclusions and Further Research

In this paper, we have presented results of an interview study performed in a large software development company in Sweden. The goal of the study was to explore the current challenges in aligning requirements and verification processes.

One of the main challenges found regards software tools, both for managing requirements and for managing test cases. Often tools are not easy to use, and when different tools are used, the interface between them is poor. The consequence is that employees tend to not update the requirements stored in the tools and the information

stored becomes obsolete and not useful. Traceability is also a challenge, and its importance is corroborated by other studies [3, 14]. Communication and cooperation across different units within the company is also a major challenge, confirming the results in [1, 17]. As a consequence of the challenges, company has decided to improve it's development process.

Our results can inspire other practitioners in their alignment improvement efforts since they can learn from this case what can be the most salient challenges in managing large quantities of requirements and test information in natural language. Researchers can also learn from this study since they can focus their research on existing challenges of potentially general interest.

We are extending this study to other companies of different size and domain. This will further enhance a general picture of alignment issues.

Acknowledgements. This work was funded by the Industrial Excellence Center EASE - Embedded Applications Software Engineering, (http://ease.cs.lth.se). Many thanks to the anonymous interviewees for their dedicated participation in this study, and reviewers of the paper for their valuable comments.

References

1. Karlsson, L., Dahlstedt, Å.G., Regnell, B., Natt Och Dag, J., Persson, A.: Requirements Engineering Challenges in Market-Driven Software Development – An Interview Study with Practitioners. Information and Software Technology 49(6), 588–604 (2007)
2. Pohl, K., Böckle, G., Van Der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
3. Post, H., Sinz, C., Merz, F., Gorges, T., Kropf, T.: Linking Functional Requirements and Software Verification. In: 17th IEEE International Requirements Engineering Conference, pp. 295–302. IEEE Computer Society, Atlanta (2009)
4. Uusitalo, E.J., Komssi, M., Kauppinen, M., Davis, A.M.: Linking Requirements and Testing in Practice. In: 16th IEEE International Requirements Engineering Conference, pp. 265–270. IEEE Computer Society, Barcelona (2008)
5. Runeson, P., Höst, M.: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering 14(2), 131–164 (2009)
6. Lubars, M., Potts, C., Richter, C.: A Review of the State of the Practice in Requirements Modeling. In: 1st IEEE International Symposium on Requirements Engineering, pp. 2–14. IEEE Computer Society, San Diego (1993)
7. Berntsson Svensson, R., Gorschek, T., Regnell, B.: Quality Requirements in Practice: An Interview Study in Requirements Engineering for Embedded Systems. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 218–232. Springer, Heidelberg (2009)
8. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Natt Och Dag, J.: An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. In: 5th IEEE International Symposium on Requirements Engineering, pp. 84–91. IEEE Computer Society, Toronto (2001)
9. Chatzoglou, P.D.: Factors Affecting Completion of the Requirements Capture Stage of Projects with Different Characteristics. Information and Software Technology 39(9), 627–640 (1997)

10. 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 Transactions on Software Engineering 32(7), 433–453 (2006)
11. Regnell, B., Berntsson Svensson, R., Wnuk, K.: Can We Beat the Complexity of Very Large-Scale Requirements Engineering? In: Paech, B., Rolland, C. (eds.) REFSQ 2008. LNCS, vol. 5025, pp. 123–128. Springer, Heidelberg (2008)
12. Seaman, C.B.: Qualitative Methods. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, ch. 2. Springer, Heidelberg (2008)
13. Nurmuliani, N., Zowghi, D., Fowell, S.: Analysis of Requirements Volatility during Software Development Life Cycle. In: 2004 Australian Software Engineering Conference, p. 28. IEEE Computer Society, Washington (2004)
14. ISO/IEC 9126 – Software and System Engineering – Product quality – Part 1: Quality model (1999-2002)
15. Fricker, S., Gorschek, T., Byman, C., Schmidle, A.: Handshaking: Negotiate to Provoke the Right Understanding of Requirements. IEEE Software (2009)
16. Robson, C.: Real World Research, 2nd edn. Blackwell, Malden (2002)
17. Jones, J.A., Grechanik, M., Van der Hoek, A.: Enabling and Enhancing Collaborations between Software Development Organizations and Independent Test Agencies. In: Cooperative and Human Aspects of Software Engineering (CHASE), Vancouver (2009)
18. Lombard, M., Snyder-Duch, J., Campanella Bracken, C.: Content Analysis in Mass Communication - Assessment and Reporting of Intercoder Reliability. Human Communication Research 28(4), 587–604 (2002)

On the Perception of Software Quality Requirements during the Project Lifecycle

Neil A. Ernst and John Mylopoulos

Department of Computer Science
University of Toronto
Toronto, ON, Canada
{nernst,jm}@cs.toronto.edu

Abstract. [**Context and motivation**] A key requirements consideration in software development is the system's quality requirements. Quality is usually defined in terms of global properties for a software system, such as "reliability", "usability" and "maintainability". In the context of software maintenance they are particularly relevant: maintenance activities are performed to ensure software quality. [**Question/problem**] Recently an expanded view of RE has been emerging, wherein requirements artifacts play a role throughout a system's lifecycle. How important are quality requirements as the lifecycle progresses? We examine two questions: whether requirements are discussed more as the software matures; secondly, whether different software projects have similar levels of interest about quality requirements. [**Principal ideas/results**] We use a software repository mining technique we call signifier extraction, and empirically investigate the treatment of software quality in software projects. Signifiers are keywords about quality requirements that we generate using a controlled taxonomy based on ISO9126. Using source data extracted from eight open-source software projects we extract the signifier frequencies over weekly intervals. We analyze the signifier occurrence patterns statistically and historically. [**Contribution**] Our results show that quality requirements are discussed differently in different projects. Furthermore, there is no correlation between project age and the importance of software quality requirements. Finally, we show that these occurrences provide a roadmap to reconstruct the historical changes of qualities as responses to external forces, such as release cycles and usability audits.

Keywords: Evolution, software quality requirements, repository mining.

1 Introduction

Software quality requirements are a key concern throughout the software lifecycle. Requirements research is increasingly focused on supporting systems beyond the initial design phase, captured by Finkelstein's term 'reflective requirements' [6]. Quality requirements are usually defined in terms of global properties for a software system, such as "reliability", "usability" and "maintainability";

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 143–157, 2010.

we think of them as describing the 'how', rather than the 'what'. In this sense "functionality" can also be considered a quality, insofar as it describes how well a given artifact implements a particular function (such as security). The importance of quality requirements lies in their inter-system comparability. Because of their global nature, quality requirements are hard to build into a design and are often treated *post facto* in terms of metrics that are applied to the final product.

If requirements are important throughout the life-cycle (and we believe strongly that they are), a better understanding of requirements after the initial release is important. Are requirements discussed post-release? One way of answering this question is to examine current practices using a standardized requirements taxonomy. In particular, we are interested in finding out whether there is any noticeable pattern in how software project participants conceive of quality requirements. Our study is conducted from the perspective of project participants (e.g., developers, bug reporters, users). We use a set of eight open-source software (OSS) products to test two specific questions about software quality requirements. The first is whether software quality requirements are of more interest as a project ages, as predicted in Lehman's 'Seventh Law' that "the quality of systems will appear to be declining unless they are rigorously maintained and adapted to environmental changes [15, p. 21]." Our second question is whether quality is of similar concern among different projects. That is, is a quality such as *Usability* as important to one project's participants as it is to another?

To assess these questions, we need to define what we mean by software quality requirements. Our position is that requirements for software quality can be conceived as a set of labels assigned to the conversations of project participants. These conversations take the form of mailing list discussions, bug reports, and commit logs. Consider two developers in an OSS project who are concerned about the software's performance. To capture this quality requirement, we look for indicators, which we call signifiers, which manifest the concern. We then label the conversations with the appropriate software quality, using text analysis. Our qualities are derived from a standard taxonomy the ISO 9126-1 software quality model [9]. The signifiers are keywords that are associated with a particular quality. For example, we label a bug report mentioning the **slow** response time of a media player with the *Efficiency* quality.

We discuss related approaches in Section 2. Section 3 describes how we derive these signifiers and how we built our corpora and toolset for extracting the signifiers. We then present our observations and a discussion about significance in Section 4. Finally, we examine some threats to our approach and discuss future work.

2 Related Work

Part of our effort with this project is to understand the qualitative and intentional aspects of requirements in software evolution, a notion we first discussed in [11]. That idea is derived from work on narratives of software systems shown in academic work like [1].

Cleland-Huang and her colleagues published work on mining requirements documents for non-functional requirements (quality requirements) in [8]. One approach they tried was similar to this one, with keywords mined from NFR catalogues found in [7]. They managed recall of 80% with precision of 57% for the Security NFR, but could not find a reliable source of keywords for other NFRs. Instead, they developed a supervised classifier by using human experts to identify an NFR training set. There are several reasons we did not follow this route. One, we believe we have a more comprehensive set of terms due to the taxonomy we chose. Secondly, we wanted to compare across projects. Their technique was not compared across different projects and the applicability of the training set to different corpora is unclear. A common taxonomy allows us to make inter-project comparison (subject to the assumption that all projects conceive of these terms in the same way). Finally, the source text we use is less structured than their requirements documents.

Massey [16] and Scacchi ([19,20]) looked at the topic of requirements in open-source software. Their work discusses the source of the requirements and how they are used in the development process. German [13] looked at GNOME specifically, and listed several sources for requirements: leader interest, mimicry, brainstorming, and prototypes. None of this work addressed quality requirements in OSS, nor did it examine requirements trends.

The difference between projects in level of interest in particular quality requirements was detailed in several case studies described in [10], which describes a methodology which starts with ISO9126 and 'tailors' the requirements analysis to specific NFRs.

3 Methodology

Overview: We first construct a set of signifiers, which produces a word list to extensionally define the software quality of interest, e.g., *Efficiency*. We then query corpora from each project with these lists to identify events. Events are timestamped occurrences of our signifiers in the corpora.

3.1 Step I Establishing the Corpora

Our corpora are from a selection of eight Gnome projects, listed in Table 1. Gnome is an OSS project that provides a unified desktop environment for Linux and its cousins. Gnome is both a project and an ecosystem: while there are co-ordinated releases, each project operates somewhat independently. In 2002, Koch and Schneider [14] listed 52 developers as being responsible for 80% of the Gnome source code. In our study, the number of contributors is likely higher, since it is easier to participate via email (e.g., feature requests) or bug reports. For example, in Nautilus, there were approximately 2,000 people active on the mailing list, whereas there were 312 committers to the source repository. [1]

[1] Generated using Libresoft, tools.libresoft.es

Table 1. Selected Gnome ecosystem products (*ksloc* = thousand source lines of code)

Product	**Language**	**Size** *(ksloc)*	**Age** *(years)*
Evolution	`C`	313	10.75
Nautilus	`C`	108	10.75
Metacity	`C`	66	7.5
Ekiga	`C++`	54	7
Totem	`C`	49	6.33
Deskbar	`Python`	21	3.2
Evince	`C`	66	9.75
Empathy	`C`	55	1.5

The projects used in this paper were selected to represent a variety of lifespans and codebase sizes (generated with [21]). All projects were written in C/C++, save for one in Python (Deskbar). For each project we created a corpus from that project's mailing list, subversion logs and the bug comments, as of November 2008. From the corpus, we extracted 'messages', that is, the origin, date, and text (e.g, the content of the bug comment), and placed this information into a MySQL database. A message consists of a single bug report, a single email message, or a single commit. If a discussion takes place via email, each individual message about that subject is treated separately. Our dataset consists of over nine hundred thousand such messages, across all eight projects. We do not extract information on the mood of a message: i.e., we cannot tell whether this message expressed a positive attitude towards the requirement in question (e.g., "This menu is unusable"). Furthermore, we are not linking these messages to the implementation in code; we have no way of telling to what extent the code met the requirement beyond participant comments.

Table 2. Qualities and quality signifiers – Wordnet version (WN). Bold text indicates the word appears in ISO/IEC 9126.

Quality	**Signifiers**
Maintainability	**testability changeability analyzability stability maintainability** maintain maintainable modularity modifiability understandability
Functionality	**security compliance accuracy interoperability suitability** functional practicality **functionality**
Portability	**conformance adaptability replaceability installability** portable movableness movability **portability**
Efficiency	**resource behaviour time behaviour** efficient **efficiency**
Usability	**operability understandability learnability** useable usable serviceable usefulness utility useableness usableness serviceableness serviceability **usability**
Reliability	**fault tolerance recoverability maturity** reliable dependable responsibleness responsibility reliableness **reliability** dependableness dependability

3.2 Step II Defining Qualities with Signifiers

In semiotics, Peirce drew a distinction between signifier, signified, and sign [2]. In this work, we make use of signifiers words like 'usability' and 'usable' to capture the occurrence in our corpora of the signified in this example, the concept *Usability*. We extract our signified, concept words from the ISO 9126 quality model [9], which describes six high-level quality requirements (listed in Table 2). There is some debate about the significance and importance of the terms in this model. However, it is "an international standard and thus provides an internationally accepted terminology for software quality [3, p. 58]," which is sufficient for the purposes of this research.

We want to preserve domain-independence, such that we can use the same set of signifiers on different projects. This is why we eschew more conventional text-mining techniques that generate keyword vectors from a training set.

We generate the initial signifiers from Wordnet [12], an English-language 'lexical database' that contains semantic relations between words, including meronymy and synonymy. We extract signifiers using Wordnet's synsets, hypernyms, and related forms (stems), and related components using the two-level hierarchy in ISO9126. When we account for spelling variations, we associate this wordlist with a top-level quality, and use that to find unique events. This gives us a repeatable procedure for each signified quality. We call this initial set of signifiers WN.

Expanding the signifiers – The members of the set of signifiers will have a big effect on the number of events returned. For example, the term 'user friendly' is one most would agree is relevant to discussion of usability. However, this term does not appear in Wordnet. To see what effect an expanded list of signifiers would have, we generated a second set (henceforth ext), by expanding WN with more software-specific signifiers. The ext signifier sets are shown in Table 3.

To construct our expanded sets, we first used Boehm's 1976 software quality model [4], and classified his eleven 'ilities' into their respective ISO9126 qualities. We did the same for the quality model produced by McCall et al. [17]. Finally, we analyzed two mailing lists from the KDE project to enhance the specificity of the sets. Like Gnome, KDE is an open-source desktop suite for Linux, and likely uses comparable terminology. We selected KDE-Usability, which focuses on usability discussions for KDE as a whole; and KDE-Konqueror, a list about a long-lived web browser project. For each high-level quality in ISO9126, we first searched for our existing (WN) signifiers; we then randomly sampled twenty-five mail messages that were relevant to that quality, and selected co-occurring terms relevant to that quality. For example, we add the term 'performance' to the synonyms for *Efficiency*, since this term occurs in most mail messages that discuss efficiency.

There are many other possible sources for quality signifiers, but for comparative purposes with the Wordnet lists, we felt these sources were sufficient.

We discuss the differences the two sets create in Section 4.

Table 3. Qualities and quality signifiers – extended version (ext). Each quality consists of terms in (a) in addition to the ones listed.

Quality	Signifiers
Maintainability	WN + interdependent dependency encapsulation decentralized modular
Functionality	WN + compliant exploit certificate secured buffer overflow policy malicious trustworthy vulnerable vulnerability accurate secure vulnerability correctness accuracy
Portability	WN + specification migration standardized l10n localization i18n internationalization documentation interoperability transferability
Efficiency	WN + performance profiled optimize sluggish factor penalty slower faster slow fast optimization
Usability	WN + gui accessibility menu configure convention standard feature focus ui mouse icons ugly dialog guidelines click default human convention friendly user screen interface flexibility
Reliability	WN + resilience integrity stability stable crash bug fails redundancy error failure

3.3 Step III Querying the Corpora

Once we constructed our sets of signifiers, we applied them to the message corpora (the mailing lists, bug trackers, and repositories) to create a table of events. An event is any message (row) in the corpus table which contains at least one term in the signifier set. A message can contain signifiers for different qualities, and can this generate as many as six events (e.g., a message about maintainability and reliability). However, multiple signifiers for the *same* quality only generate a single event for that quality. We produced a set of events (e.g., a subversion commit message), along with the associated time and project. We group events by week for scalability reasons. Note that each email message in a thread constitutes a single event. This means that it is possible that a single mention of a signifier in the original message might be replied to multiple times. We assume these replies are 'on-topic' and related to the original concern.

We normalize the extracted event counts to remove the effect of changes in mailing list volume or commit log activity (some projects are much more active). The calculation takes each signifier's event count for that period, and divides by the overall number of messages in the same period. We also remove low-volume periods from consideration. This is because a week in which only one message appeared, that contained a signifier, will present as a 100% match. From this dataset we conducted our observations and statistical tests. Table 4 illustrates some of the sample events we dealt with, and our subsequent mapping to software quality requirements.

3.4 Step IV Precision and Recall

We verified the percentage of terms retrieved that were unrelated to a signified software quality to understand the precision of our method. For example, we

Table 4. Classification examples. Signifiers causing a match are highlighted.

Event	**Quality**
...By upgrading to a newer version of GNOME you could receive **bug** fixes and new functionality.	*None*
There should be a feature added that allows you to keep the current **functionality** for those on workstations (automatic hot-sync) and then another option that allows you to manually initiate .	*Functionality*
Steps to reproduce the **crash**: 1. Can't reproduce with **accuracy**. Seemingly random.	*Reliability, Functionality*
How do we go disabling ekiga's **dependency** on these functions, so that people who arn't using linux can build the program without having to resort to open heart surgery on the code?	*Maintainability*
U_() is equivalent of _() but returns **Unicode** (UTF-8) string. Update your xml-**i18n**-tools from CVS (recent version understands U_), update Swedish **translation** and close the bug back.	*Portability*
On some thought, centering **dialogs** on the panel seems like it's probably right, assuming we keep the **dialog** on the screen, which should happen with latest metacity.	*Usability*
These calls are just a waste of time for client and server, and the Nautilus online storage view is **slowed** down by this wastefulness.	*Efficiency*

encountered some mail messages from individuals whose email signature included the words "Usability Engineer". If the body of the message wasn't obviously about usability, we coded this as a false-positive. Our error test was to randomly select messages from the corpora and code them as relevant or irrelevant. We assessed 100 events per quality, for each set of signifiers (ext and WN). Table 5 presents the results of this test. False-positives averaged 21% and 20% of events, for ext and WN respectively (i.e., precision was 79% and 80%).

Recall, or completeness, is defined as the number of relevant events retrieved divided by the total number of relevant events. Superficially we could describe our recall as 100%, since the query engine returns all matches we asked for, but true recall should be calculated using all events that had that quality as a topic. To assess this, we randomly sampled our corpora and classified each

Table 5. False positive rates for the two signifier sets

Signified quality	F.P. Rate ext	F.P. Rate WN
Usability	0.47	0.22
Portability	0.11	0.20
Maintainability	0.22	0.31
Reliability	0.15	0.19
Functionality	0.14	0.18
Efficiency	0.16	0.07
Mean	**0.21**	**0.20**

event into either a signifier (*Usability*, *Reliability*, etc.) or *None*. For extended signifier lists, we had an overall recall of 51%, and a poor 6% recall for the Wordnet signifiers. We therefore dispensed with the Wordnet signifiers. This is a very subjective process. For example, we classified a third of the events as *None*; however, arguably any discussion of software could be related, albeit tangentially, to an ISO9126 quality. We think a better understanding of this issue is more properly suited to a qualitative study, in which project-specific quality models can be best established.

4 Observations and Discussion

This section first explains the frequency distributions of the data we collected. We then use that data to answer the two questions raised in the introduction: 1) Is there a correlation between discussion of quality requirements and project age? 2) Are quality requirements of similar importance relative to each project?

4.1 Data Distribution

Fig. 1 shows an example frequency distribution for the quality requirement *Usability*, product *Evolution*, with non-normalized data. The distributions seem to follow a power-law distribution, that is, a majority of weeks had few events, with the 'long tail' consisting of those weeks with many events. We verified that this pattern also existed for the remaining qualities and project combinations.

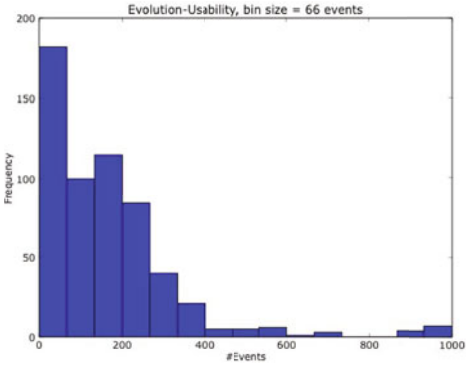


Fig. 1. Frequency distribution for Evolution-Usability. *x-axis* represents number of events (66 events wide), *y-axis* the number of weeks in that bin.

4.2 Examining Quality Discussions over Time

Our first question was whether, as predicted in the literature, there was a correlation between the importance of software quality requirement and the age of a project. We examined this in three ways. First, we looked at the overall trends for a project. Secondly, we used release windows, the time between the release

Table 6. Selected summary statistics, normalized. Examples from Nautilus and Evolution for all qualities using extended signifiers.

Project	Quality	r^2	slope	N (weeks)
Evolution	Efficiency	0.06	-0.02	439
	Portability	0.08	-0.05	448
	Maintainability	0.04	-0.02	320
	Reliability	0.20	0.25	492
	Functionality	0.03	-0.02	439
	Usability	0.14	0.27	515
Nautilus	Efficiency	0.16	-0.10	420
	Portability	0.16	-0.07	331
	Maintainability	0.27	-0.09	216
	Reliability	0.19	0.26	454
	Functionality	0.12	-0.05	390
	Usability	0.08	0.29	459

of one version, and the release of the next (major) version. Finally, we explored qualitative explanations for patterns in the data.

Using project lifespan – We examined whether, over a project's complete lifespan, there was a correlation with quality event occurrences. Recall that we define quality events as occurrences of a quality signifier in a message in the corpora. We performed a linear regression analysis and generated correlation coefficients for all eight projects and six qualities. Figure 2 is an example of our analysis. It is a scatterplot of quality events vs. time for the *Usability* quality in Evolution. For example, in 2000/2001, there is a cluster around the 300 mark, using the extended (`ext`) set of signifiers. Note that the y-axis is in units of (events/volume * 1000) for readability reasons.

The straight line is a linear regression. The dashed vertical lines represent Gnome project milestones, with which the release dates of the projects we study are synchronized. Release numbers are listed next to the dashed lines. Due to

Table 7. Selected summary statistics, normalized. Examples from *Usability* and *Efficiency* (performance) for selected products using extended signifiers.

Quality	Project	r^2	slope	N (weeks)
Usability	Deskbar	0.08	-0.97	126
	Evolution	0.14	0.27	515
	Nautilus	0.08	0.29	459
	Totem	0.20	0.63	314
Efficiency	Deskbar	0.00	-0.11	34
	Evolution	0.06	-0.02	439
	Nautilus	0.16	-0.10	420
	Totem	0.10	-0.16	158

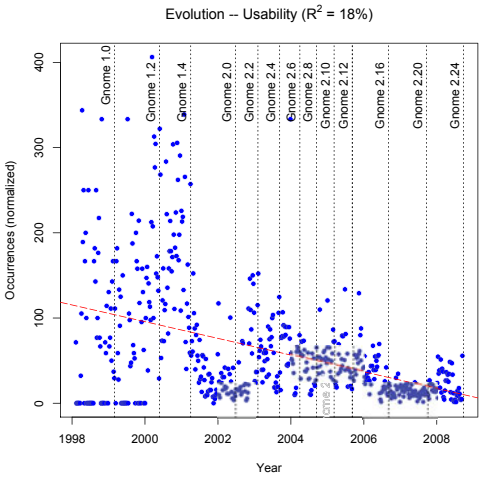


Fig. 2. Signifier occurrences per week, Evolution – *Usability*

space constraints, Table 6 lists only Nautilus and Evolution as products, and r^2 squared correlation value, or coefficient of determination and slope (trend) values for each quality within that project. r^2 varies between 0 and 1, with a value of 1 indicating perfect correlation. The sign of the slope value indicates direction of the trend. A negative slope would imply a decreasing number of occurrences as the project ages. Table 7 does a similar analysis for all products and the *Usability* and *Efficiency* (performance) qualities.

The results are inconclusive. In all cases the correlation coefficient indicating the explanatory power of our linear regression model is quite low, well below the 0.9 threshold used in, for example, [18]. There does not seem to be any reason to move to non-linear regression models based on the data analysis we performed. We conclude that our extended list of signifiers does not provide any evidence of a relationship between discussions of software quality requirements and time. In other words, either the occurrences of our signifiers are random, or there is a pattern, and our signifier lists are not adequately capturing it. The former conclusion seems more likely based on our inspection of the data.

Using release windows – It is possible that the event occurrences are more strongly correlated with time periods prior to a major release, that is, that there is some cyclical or autocorrelated pattern in the data. We defined a release window as the period from immediately after a release to just before the next release. We investigated whether there was a higher degree of correlation between the number of quality events and release age, for selected projects and keywords. Was this release window correlation better than the one we found for project lifespan as a whole? For space reasons we do not include these results, but there was no improvement in correlation. There is no relationship between an approaching release date and an increasing interest in software quality requirements.

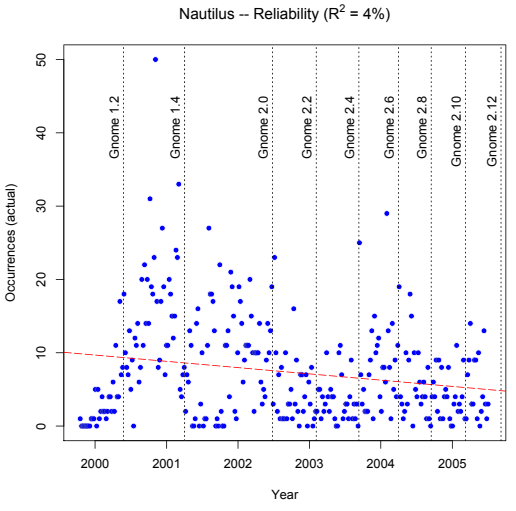


Fig. 3. Signifier occurrences per week, Nautilus – *Reliability*

Analysis of key peaks in selected graphs – The final explanation we explore is that the data are unrelated to software age or release cycle, and are instead responding to external events, such as a usability audit. We chose to look at Evolution, a mail and calendar client, and Nautilus, a web browser and file manager, for more detailed 'historical' analysis. We tried two approaches: one used the normalized data, and identified periods where our signifier occurred more frequently with respect to everyday volume. The second approach used the actual signifier counts to see why that signifier occurred more frequently than other periods.

We looked at the normalized *Usability* events in Evolution, shown in Fig. 2. To eliminate bug reports and triaging events, we excluded these types of data from our query. Many bug reports are auto-generated, and contribute more noise than signal. For instance, one initial peak we examined was related to the "Mass close of stale bugs ¿ = 4 months old." This generated a lot of noise as the signifiers in these reports are considered once more by our algorithm (since we treat any discussion on a bug similarly to mail threads).

With these events removed as noise, Fig. 2 shows a cluster of points in early 2000. Mailing list discussions at that time turned to a question about the default option for forwarding mail messages, e.g., "... I know this was discussed a few weeks ago ... could it be implemented as an advanced option that has to be turned on and is off by default?" Later that year, in October, another spike in our graph can be attributed to a feature-freeze on Evolution and associated UI cleanups. As Evolution 1.4 is released in mid-2003, there is a small upward trend. Events at that time reflect problems with the new release, reflecting some UI changes. We still see some effects due to volume, such as the outlier near the end of 2003, where nearly one third of mailing messages were usability related. The issue here is one of overall volume over the winter holidays. In this case a single mail thread about keyboard shortcuts consumed the discussions.

For our second approach, we used the actual signifier event counts, and targeted *Reliability* events for Nautilus. In November, 2000, 50 events occur. Inspecting the events, one can see that a number have to do with bug testing the second preview release that was released a few days prior. For example, one event mentions ways to verify reliability requirements using hourly builds: "As a result, you may encounter a number of **bugs** that have already been fixed. So, if you plan to submit **bug** reports, it's especially important to have a correct installation!". Secondly, in early 2004 there is a point with 29 events just prior to the release of Gnome 2.6. Discussion centers around the proper treatment of file types that respects reliability requirements. It is not clear whether these discussions are in response to the external pressure of the deadline or are just part of a general, if heated, discussion.

These investigations show that there is value to examining the historical record of a project in detail, beyond quantitative analysis. While some events are clearly responding to external pressures such as release deadlines, other events are often prompted by something as simple as participant interest, which seems to be central to the OSS development model.

4.3 Quality Importance and Project

Recall that in our second question, we wanted to examine whether certain projects would be more concerned with software quality requirements than others. We characterized the importance of a requirement to a project by calculating the mean normalized occurrences of the signifier (such as *Usability*) over time. This controls for both project longevity and project size.

Table 8 lists our results; for space considerations, only three (representative) qualities are listed. We show the mean number of occurrences per week, normalized by dividing by the overall number of 'events' in that period, to eliminate the effect of volume. We would like to know, in other words, what proportion of all messages in that week were talking about the requirement of interest.

We used the extended signifier set (`ext`). We cannot compare between qualities, because the signifier sets are not the same size. However, there is a difference among projects. We chose to focus on Nautilus and Evolution (both projects of similar longevity, focused on file management and mail respectively). The

Table 8. Quality per project. Numbers indicate normalized occurrences per week.

Quality	Project	Occurrences
Efficiency	Evolution	**0.012**
	Nautilus	**0.026**
Usability	Evolution	**0.192**
	Nautilus	**0.285**
Portability	Evolution	**0.010**
	Nautilus	**0.011**

Efficiency quality occurs in Nautilus discussions at a rate of 0.026 occurrences per week, and in Evolution at 0.012 occurrences per week less than half as often. *Usability* is discussed 1.5 times as often in Nautilus, while other requirements, including *Portability*, show no difference. One possible explanation is that Evolution participants have a conceptual model of Efficiency that is a poorer fit to our signifier lists than the model Nautilus participants use. However, it does seem fair to conclude that projects have different interests with respect to software quality. We intend to do further testing to explore how communities conceptualize these fairly abstract '-ilities'.

4.4 Threats to Validity

Construct validity. The main threat to construct validity is that our signifiers may omit relevant terms or phrases., e.g., "can't find the submit button" vs. "usability". Our qualities are not directly comparable, since their respective signifier set sizes differ. *Usability*, for example, has 24 terms in its bubble, versus *Functionality* with 10. We conducted the error analysis to determine how accurate our bubbles are. Our error validation should be conducted by more people to ensure inter-rater reliability. Many events are tricky to classify. Furthermore, we are assuming that projects share the ontology of software quality expressed in the quality model (ISO9126). A more domain-specific taxonomy would be useful.

Internal validity. When we perform our regression analysis, assuming a linear relationship may not be a good model of the actual pattern these discussions follow. We focused on the linear model as it is the simplest explanation of the pattern we would expect to see if quality discussions were increasing with time. Our source data may not capture all discussions regarding quality requirements – we omitted IRC chats, for instance. However, these data sources are most amenable to large-scale analysis. Follow-up with qualitative studies would be useful.

External validity. Our data originated from open-source projects, less than ten years old, from the Gnome ecosystem. Of these, the open-source nature of the project seems most problematic for external validity. Capra et al. [5], for example, show a higher software quality in OSS projects than commercial projects. It would be interesting to determine whether a top-down directive to focus on software quality, or some other methodological change, would present as a noticeable spike on the event occurrence graph.

4.5 Models of Quality Requirements

There is a rich history of discussion regarding software quality requirements, and quality models in particular. The main problem that arose in our study was that the quality models are (by design) very high-level. They provide a useful baseline from which to derive more specific models. However, it is useful to have a cross-product quality requirements model which can be used to compare software systems. Many questions are left unanswered when confronted by actual data:

for instance, what is the relationship between product reliability and product functionality?

The challenge for researchers is to align software quality models, at the high level, with the product-specific requirements models developers and community participants work with, even if these models are implicit. One reason discussions of quality requirements were difficult to identify is that, without explicit models, these requirements are not properly considered or are applied haphazardly. We need to establish a mapping between the platonic ideal and the reality on the ground. This will allow us to compare maintenance strategies for product quality requirements across domains, to see whether strategies in, for example, Gnome, can be translated to KDE, Apple, or Windows software.

5 Conclusions and Future Work

This paper presents a novel analysis technique for conducting empirical research in Requirements Engineering. The technique has been applied to study two specific questions concerning quality requirements. In accordance with Lehman's laws of software evolution, we hypothesized that there is growing interest in quality requirements within a developer community as a project matures. However, our analysis provides no evidence for this hypothesis. However, it is sometimes possible to use external events to explain patterns in the data. We then showed that there is a difference in how different projects treat software qualities with some projects discussing certain quality requirements more than others. We have not presented a formal hypothesis about what this might suggest.

Our ultimate goal is to be able to extract, from available sources, a list of requirements for a project, so that we can trace not just the 'physical' changes in the codebase, but also the evolving features and goals inherent in a project. We plan to continue our experiments with repository mining with this in mind. We have begun work using multi-label classifiers on more domain-specific taxonomies (e.g., database systems). We think 'ground-truthing' our results with qualitative studies would be useful to make our results inform a theory about quality in software, such that techniques could be predictive as well as descriptive.

Appendix and Acknowledgements

We appreciate the comments of the software engineering group at the University of Toronto and Abram Hindle, and the comments of anonymous reviewers. Source code, processed data, and related discussions are available at http://neilernst.net/tag/msr/.

References

1. Antón, A.I., Potts, C.: Functional paleontology: system evolution as the user sees it. In: International Conference Software Engineering, Toronto, Canada, pp. 421–430 (2001)

2. Atkin, A.: Peirce's Theory of Signs (October 2006), http://plato.stanford.edu/entries/peirce-semiotics/
3. Bøegh, J.: A New Standard for Quality Requirements. IEEE Soft. 25(2), 57–63 (2008)
4. Boehm, B., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality. In: International Conference Software Engineering, pp. 592–605 (1976)
5. Capra, E., Francalanci, C., Merlo, F.: An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects. Trans. Soft. Eng. (2008)
6. Cheng, B.H., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
7. Chung, L., Nixon, B.A., Yu, E.S., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol. 5. Kluwer Academic Publishers, Boston (October 1999)
8. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In: International Conference Requirements Engineering, pp. 39–48. IEEE, Minneapolis (2006)
9. Coallier, F.: Software engineering – Product quality – Part 1: Quality model. Tech. Rep. ISO9126, International Standards Organization - JTC 1/SC 7 (2001)
10. Doerr, J., Kerkow, D., Koenig, T., Olsson, T., Suzuki, T.: Non-Functional Requirements in Industry - Three Case Studies Adopting an Experience-based NFR Method. In: International Conference Requirements Engineering, pp. 373–384 (2005)
11. Ernst, N.A., Mylopoulos, J.: Tracing software evolution history with design goals. In: International Workshop on Software Evolvability at ICSM, Paris, France (October 2007)
12. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
13. German, D.M.: The GNOME project: a case study of open source, global software development. Soft. Process: Improvement and Practice 8(4), 201–215 (2003)
14. Koch, S., Schneider, G.: Effort, co-operation and co-ordination in an open source software project: GNOME. Inf. Sys. J. 12, 27–42 (2002)
15. Lehman, M.M., Ramil, J.F., Wernick, P.D., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution-the nineties view. In: Int. Soft. Metrics Symp., Albuquerque, NM, pp. 20–32 (1997)
16. Massey, B.: Where Do Open Source Requirements Come From (And What Should We Do About It)? In: Workshop on Open Source Software Engineering at ICSE, Orlando, FL, USA (2002)
17. McCall, J.: Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisition Manager. General Electric, vol. 1-3 (November 1977)
18. Mens, T., Fernandez-Ramil, J., Degrandsart, S.: The evolution of Eclipse. In: International Conference Software Maintenance, Shanghai, China, pp. 386–395 (October 2008)
19. Scacchi, W.: Understanding the requirements for developing open source softwaresystems. IET Software 149(1), 24–39 (2002)
20. Scacchi, W., Jensen, C., Noll, J., Elliott, M.: Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. In: International Conference on Open Source Software, Genoa, Italy, vol. 1, pp. 1–8 (July 2005)
21. Wheeler, D.: SLOCcount (2009), http://www.dwheeler.com/sloccount/

Lessons Learned from Integrating Specification Templates, Collaborative Workshops, and Peer Reviews

Marko Komssi[1,2], Marjo Kauppinen[1], Kimmo Toro[2], Raimo Soikkeli[3], and Eero Uusitalo[1]

[1] Software Business and Engineering Institute, Aalto University School of Science and Technology, P.O. Box 9210, 02015 TKK, Finland
[2] F-Secure Corporation, Finland
[3] Ilmarinen Mutual Pension Insurance Company, Finland
{Marko.Komssi,Marjo.Kauppinen,Eero.Uusitalo}@tkk.fi,
Kimmo.Toro@f-secure.com, Raimo.Soikkeli@ilmarinen.fi

Abstract. [Context & motivation] Specifying requirements and ensuring their quality are critical for the success of software development projects. A variety of practices have been suggested to manage these activities, including specification templates, collaborative workshops, and peer reviews, but few empirical studies exist on their inter-connection. [Question/problem] We studied the lessons learned from integrating these three practices: "what are the problems faced with the use of the three practices?" and "what kind of approach supports the integrated use of the three practices?" [Principal ideas/results] In the Finnish companies included in the study, the key problems with the use of the three practices seemed to be the following: 1) the use of the three practices was typically inadequately established to meet the needs of the particular projects and 2) the requirements were communicated to key participants late and insufficiently. To avoid these types of problems, it was found useful to use the practices in an integrated manner, so that the forms of collaboration between key participants were determined and the appropriate types of practices were selected and tailored in a project-specific way. [Contribution] The paper describes the success factors of the integration. The setup workshop is introduced to support the tailoring and integration of the practices.

Keywords: Requirements engineering, setup workshop, action research, industrial experience, integration, specification, quality control, best practices.

1 Introduction

Requirements engineering (RE) is a central part of software development, but despite this fact, actual knowledge of the RE process is lacking [1]. A large part of RE research concentrates on methods or techniques supporting a single activity instead of promoting an integrated view of RE and lacks reports on the connections between various good practices [1, 2].

Several best RE practices have been identified that contribute to software project success [1]. However, having the best RE practices in place may not be enough. A best RE practice may include various techniques. The proper selection and tailoring of

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 158–172, 2010.

a technique is typically needed in order to find an appropriate fit to a particular software project that has its own characteristics [3, 4]. Unfortunately, most RE literature does not sufficiently describe the suitable target areas or limitations of specific techniques [3]. Moreover, the lack of empirical research in method tailoring is surprising in the applied field of software engineering [4].

According to Katasonov and Sakkinen [2], requirements quality control is mostly a matter of communicating requirements. They emphasize that requirements are not discovered but constructed, and there is often some disagreement between stakeholders about goals. They argue that requirements quality control cannot be seen simply as a mechanical process of checking documents but should instead be studied as a coherent entity. Moreover, to overcome the deficiencies of requirements documents, communication links between stakeholders and requirement owners are needed [5].

This paper focuses on linking three practices of RE – specification templates, collaborative workshops, and peer reviews – to support a coherent view of specifying requirements with quality control. Our industrial research study was based on the following question: what lessons have been learned from integrating these three practices? In order to answer the question, an action research study was conducted in five companies that have applied the practices in their software projects.

Drawing on the lessons learned, this paper presents an approach to integrating the above-mentioned three practices. This approach allows the practices to be tailored to fit the operating environment of a software project and form a coherent whole that enables the practices to overcome each other's deficiencies. A key part of the approach is the setup workshop, where the application and timing of these practices is analyzed and decided upon collaboratively.

The paper is organized as follows. Section 2 presents the related work from the three practices and Section 3 explains the research design. Section 4 presents the problems faced with the use of the three practices. Section 5 presents the success factors of the integration ands proposes the setup workshop as a crucial component of the integration. Finally, Section 6 concludes the findings with suggestions for future research.

2 Related Work

It is important first to find out the strengths and weaknesses of individual practices in order to understand how these practices can work together. In this section, the previously reported characteristics of specification templates, collaborative workshops, and peer reviews are presented.

2.1 Specification Templates

The use of specification templates is a common practice that supports, in particular, the early phase of specification activity in software projects. Several specification templates, such as IEEE Recommended Practice for Software Requirements Specifications [6] and the Volere Requirements Specification Template [7], are available to

provide a comprehensive structure for documentation. These templates contain predefined sections with instructions and examples as the basis for document writing.

Successful teams have frequently transformed specification templates and examples from previous projects into rational RE activities [1]. In this study, the specification templates were identified as one of the best practices in RE. Specification templates have also been identified as one of the top ten RE guidelines and recommend their implementation in all organizations [8].

Although the use of specification templates is accepted as one of the best RE practices and most of the software companies apply specification templates in their software projects, the literature seems to provide surprisingly few empirical results from the use of specification templates in software projects. The benefits and strengths of specification templates are mainly introduced in RE books. The specification templates are intended to act as a guide to essential content and to help requirements analysts determine what belongs in the specification [7, 9]. The specification templates should contribute to documents that have a higher quality and lower cost [8]. Furthermore, the use of specification templates promotes consistent communication and helps software practitioners ask questions that they might otherwise ignore [10]. Peer reviewers can also capitalize on software templates that relate to reviewing work [8].

A few disadvantages and weaknesses of specification templates have been presented. One limitation is that they are not scalable for all types of projects. In other words, the templates often lack the flexibility that is required to respond to variability in the intended readership of a document [11]. Another drawback with specification templates is that they can lead to the production of specifications that are superficially attractive but limited in their content [12].

Some suggestions have been made about ways to improve the usefulness of specification templates. For example, it has been proposed that a suitable specification template with embedded guidance texts should be defined for each project type [10] or, similarly, that software practitioners should predefine a set of situation types and suggest an appropriate requirements document style for each type [13]. In addition, in order to develop a useful structure for a specification template, the existing documents of the organization should be investigated and ideas should be collected from document users [8].

2.2 Collaborative Workshops

The use of collaborative workshops seems to be a less common RE practice in software projects than the use of specification templates or peer reviews [1]. Collaborative workshops can vary from informal to formal and from short to long and can be used for several purposes to support different RE activities. In particular, a collaborative workshop can provide an efficient, controlled, and dynamic setting where the participants can quickly elicit, prioritize, and agree on a set of project requirements [14].

One formal approach to specifying requirements is the collaborative construction of the entire requirements document. For example, the RaPiD7 collaborative document authoring technique has been developed at Nokia [15]. In this technique, a document is created by a team in consecutive workshops, which reduces the risk of the document's content being based primarily on the judgment of the author. The RaPiD7 technique is supposed to enhance the communication and commitment of a

project team and improve document quality. The reason for these advantages is the early involvement of the team in documentation work. RaPiD7 speeds up the document creation process in terms of calendar time.

RaPiD7 is similar to a more widespread method called Joint Application Development (JAD), which was developed at IBM in 1977 [16]. JAD enables technical and business specialists to learn about each other's domain knowledge, improves communication among interested parties, facilitates consensus management, and increases user acceptance of specifications [17].

One challenge these two techniques present is the common time required for workshop meetings. For example, a JAD procedure typically lasts for three to five days [18] and stakeholders have difficulty allocating common time. In fact, it was necessary to adapt the JAD technique in some organizations because the staff were sometimes unable or unwilling to commit to full-time participation in JAD workshops [19].

Cockburn [20] proposes a more informal process for a collaborative workshop, in which people work in a full group when there is a need to align or brainstorm and use the rest of their time in pairs or alone. Cockburn explains that a group is able to brainstorm and reach a consensus effectively, but when the group is split, more text is produced.

2.3 Peer Reviews

Peer reviews are a core practice of requirements quality control [2, 21]. A peer review consists of someone other than the author of a document examining it in order to discover defects and identify improvement opportunities [22]. Eventually, a software development organization may need to acquire deeper knowledge from the types, formalities, and feasibilities of peer reviews. In particular, mature software development organizations are advised to develop capability to determine what types of peer reviews are conducted and to tailor the peer reviews in the organizations' software projects [21]. Wiegers has defined several review techniques and types, both formal and informal [23], which are listed below using his definitions.

The most formal technique of a review, *inspection*, has several characteristics that distinguish it from other review techniques. For example, a trained moderator leads meetings and co-operates with a trained team. The moderator defines the goals, collects quality data, and distributes results using a reporting process.

A *team review* is slightly more informal and imprecise than an inspection. Team reviews concentrate more on detecting defects than preventing them. A team review may be chosen if no trained inspection leaders are available.

In a *walkthrough*, the author of the document explains it to colleagues and asks for their feedback. This review type is generally informal and does not involve data collection and reporting. However, the process steps and the role of each participant may be clearly defined.

In a *passaround*, the author of the document sends it to several colleagues and gathers their feedback. The passaround technique is useful, for instance, for obtaining ideas and corrections for a new project plan.

In a *peer deskcheck*, only one checker examines the document. While this review technique requires the smallest amount of resources, it is only appropriate for products that do not have very high quality expectations or are not to be reused.

In an *ad hoc review*, the author of the program presents a problematic part of the design to a fellow worker and asks for help. Although quite informal, this review type is useful for short and tricky cases.

A team can identify the strengths and weaknesses of the review types [23]. The purpose of this is to select the proper review type for each case with regard to the organizational culture, time constraints, and business objectives. In particular, the team is advised to select the least expensive review type that fulfills the objectives of the review [22].

3 Research Design

The goal of this study is to present lessons learned from the integration of specification templates, collaborative workshops, and peer reviews. The study was conducted using an action research approach in five Finnish companies. The data were collected from the case study companies during a period of ten years (1999 to 2009) and analyzed iteratively in three phases.

3.1 Research Approach

In order to gain a deep understanding of the three practices and their integration, we applied an action research approach. This research method was selected for two reasons: it has a unique ability to link research to practice, and as a qualitative method, it is also effective for explaining what is happening in a company [24]. The action research approach allows researchers to address complex real-life matters and study selected issues in detail [25]. Additionally, an "industry-as-laboratory" research approach, where researchers identify problems through close involvement with industrial projects and create and evaluate practices addressing the problem, is suggested in [26]. This lets researchers emphasize what people actually do or can do in practice, rather than what is possible in principle.

To access insider and historical data, as well as to engage practitioners in research, we also applied the insider action research approach [27]. In the insider action research approach, some of the researchers are internal members of practitioner organizations. As internal members of the organization, practitioner-researchers have the opportunity to collect data that are richer than what they would collect as external researchers. Gummesson [28] points out that a lot of information is stored in the minds of practitioners, who have often undergone central and dramatic changes. Therefore, Gummesson urges practitioners to act as researchers and reflect on what they had learned retrospectively.

3.2 Case Study Companies

Our research was conducted in five Finnish companies, which are introduced in Table 1. Three of the companies were of medium size, one was small, and one was large. Companies A, C, and E are internationally known and have a significant global market share in their fields. These three companies focus mainly on solutions developed for a large number of customers. Company B provides pension insurance

Table 1. Case study companies

Company	Number of employees	Application domain
A	700	Computer and information security solutions for companies and consumers
B	600	Earnings-related pension insurance services
C	500	Information management systems for buildings, public infrastructure, and energy distribution designers
D	50	Language technologies and services for companies and consumers
E	24000	Transportation systems and services for buildings

services in Finland, and Company D offers language technology solutions, mainly to large companies.

Table 2 summarizes the practices the case study companies have applied. Each of the companies defined at least one specification template for requirements. In order to shorten the requirements documents, some of the organizations opted for two templates – one for high-level requirements and one for more detailed requirements. We were able to gather data related to collaborative workshops from two companies, even though they were practiced in all of the companies to some extent. In particular, Company B had a long tradition of using workshops to define requirements collaboratively. All of the case study companies had applied some kind of peer reviews, with the most common being team reviews.

Table 2. Practices investigated in each company

	A	B	C	D	E
Specification templates	X	X	X	X	X
Collaborative Workshops	X	X			
Peer Reviews	X	X	X	X	X

3.3 Data Collection and Analysis

Our study was based on the following question: "what are the lessons learned from the integration of these three practices?" Applying the "industry-as-laboratory" research approach [26], we divided the question into two more specific questions as follows: "what are the problems faced with the use of the three practices?" and "what kind of approach supports the integrated use of the three practices?" Figure 1 illustrates the three main phases of the study: 1) identification of problems faced in software projects and development of the integration approach, in Company A, 2) retrospective analysis of the problems faced in the five companies and refinement of the integration approach, and 3) validation of findings and refinement of the integration approach, in Company B.

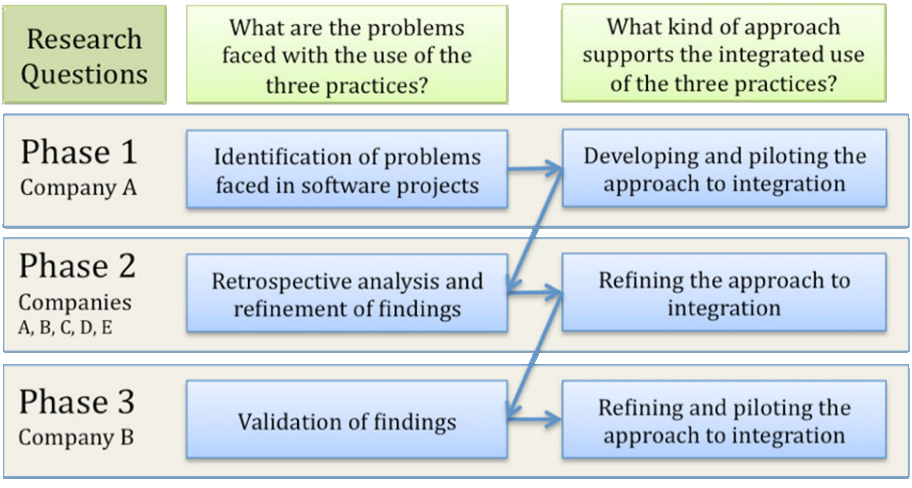


Fig. 1. Three research phases of the study

Phase 1 was performed in Company A between the years 2003 and 2006. Perceived problems of eleven software projects were first identified, and the related improvement ideas were collected and analyzed. Subsequently, a preliminary approach to the integration of the three RE practices was developed and piloted iteratively.

The goal of Phase 2 was to compare the preliminary results gained from Company A during Phase 1 with the experiences from the other four companies (B, C, D, and E). In this phase, retrospective analysis was used to examine previously collected data. The data had been collected in three ways. First, four of the authors had worked in one or two of the case study companies (A, B, D, and E), participating in software development projects and requirements process improvement work. Second, the authors conducted two research projects with the case study companies during 1999-2005. Within these research projects, data from Companies A, C, and E was gathered. Third, the authors interviewed a person who had been in charge of specification templates and peer reviews in Company A ten years ago. Based on the analysis, the authors refined the findings related to the problems faced with the use of the three practices and reflected on the approach to integration.

Phase 3 was conducted with Company B from 2006 to 2009. The company's goal was to develop a consistent yet tailorable set of RE practices that could be applied company-wide. This was accomplished by a series of 12 workshops and 19 meetings, where the findings of the previous phases were built on. The result of these activities was an approach that enables an organization to tailor and integrate specification templates, collaborative workshops, and peer reviews into a coherent entity. The approach was piloted during its development in 9 software projects. In addition, the company organized two training sessions, in which 28 project managers, requirements specialists, and group leaders participated. After the training, we asked for participants' comments on how suitable they perceive the approach as being for the types of projects they typically participate in. We used both a feedback form and group discussion to collect the data. In this phase, the collected data were analyzed and the

findings were clustered into the previous findings iteratively. The findings were validated and new findings were merged with them. The final findings are described as the lessons learned. These lessons are described in the following sections.

Table 3 summarizes the data collection activities performed in the case study companies. The results of this study are based on the data collected through observations, formal semi-structured interviews, informal conversations, the analysis of requirements specification templates, the analysis of requirements documents, and questionnaires.

Table 3. Data collection activities of the study

	A	B	C	D	E
Observation	X	X	X	X	X
Interviews	X	X	X		X
Informal conversations	X	X	X	X	X
Analysis of specification templates	X	X	X	X	X
Analysis of requirements documents	X	X	X	X	X
Questionnaires	X		X		X

3.4 Threats to Validity

We apply the explanations of Yin [29] to construct and external validity. In our study, a threat to construct validity is the possibility that we were not able to correctly collect and evaluate the problems related to the use of the three practices and the benefits and success factors from applying the integration approach. As a result, our inferences concluded as lessons learned might not represent reality, in the companies. The threat to external validity is the possibility that lessons we have learned cannot be generalized to other software development organizations.

To reduce the threat to construct validity of the study, we used of multiple sources of evidence and triangulation. We used a number of information sources and data collection techniques. We applied triangulation of data sources and data collection techniques by utilizing interviews, informal conversations, participant observation, and document analysis. In addition, the study covers a long period of time that improves the construct validity of our findings, as it was possible to analyze and validate the findings at different times. Finally, key informants from two companies reviewed our findings several times.

To reduce the threat to external validity of the research results, the study involved five separate case study organizations of different characteristics, such as size, solutions, and business environments. The integration approach was developed and piloted in two companies that have very different types of software development and business environments and solutions.

4 Problems Faced with the Use of the Three RE Practices

The use of specification templates fosters individual specification work. The application of specification templates was quite common in the companies. Software

developers who used a specification template typically specified a document alone, which tended to lead to the writing of overly comprehensive specifications. Software developers often spent a lot of time on the writing phase when using a specification template. A significant problem was that they focused too much on writing requirements with high volatility. In addition, specification templates were occasionally found to cause superficially attractive specifications that satisfied guidelines but included irrelevant content. For example, interviews and informal conversations with practitioners in a case study company revealed to us that they, as document readers, had often met and disregarded long documents whose readership was unclear. The problem was identified as a result of an individual specifying work that was based on a specification template.

The timing of a peer review is typically too late. Several reviewers were dissatisfied with the fact that participating in a review of the requirements document was often their first involvement in the development of a software system or product. The peer review usually took place once the writer felt the document was, to all intents and purposes, finished. However, the late timing of reviews was often harmful for both the writer and the reviewers. Writers were often opposed to suggestions for major changes at that point and the reviewers' interest in suggesting changes for the document seemed to decrease as well. One company employee said, "Typically, the content of the document is so refined that I do not even dare to raise any issues in the review." In practice, documents that were too polished reduced reviewers' motivation to suggest changes, even if the reviewers had discovered several weaknesses in the documents' contents.

The difference between collaborative workshops and peer reviews is not clear in practice. Even if peer reviews were originally introduced to discover defects in the documents, the practitioners of three of the case study companies gradually began to use peer reviews more for sharing information and solving problems. The reasoning for this is as follows: For some key stakeholders, such as software testers and technical support personnel, a late peer review was often the first exposure to the project's requirements. Hence, the needs of these stakeholders were not aligned with the original intent of discovering defects, but rather gaining understanding or expressing their own views on the requirements. On the other hand, some collaborative workshops that the project teams held in the late phases of specifying requirements included aspects of peer reviews; in these projects, the teams skipped the actual peer reviews. Hence, the purpose of the two activities changed or became vague. The main difference seemed to be the timing. The term "collaborative workshop" was often used in the early phase and the term "peer review" in the late phase of specifying requirements.

The purpose and content of collaborative workshops and peer reviews are poorly communicated. Practitioners in one case study company were applying peer reviews for three different purposes, depending on the degree of completion of the documents. The purposes were to obtain comments from the domain experts, to share information with several stakeholders, and to achieve acceptance of the document. However, the fact that the practitioners referred to each of the three different types of sessions as 'reviews' caused confusion. In particular, the reviewers were irritated when the

meeting was different than they had expected or wanted. In another company, reviews were part of the development process and the participants typically perceived the reviews as little more than a rubber stamp at the end of the software development procedure. On the other hand, collaborative workshops were not a defined practice in the case study companies. Undefined workshops were applied in one company and this was identified as a reason for the frustration of the participants. In particular, the goals of the workshops were not communicated and this meant that the participants had expectations of the course of action and outcomes of workshops that differed from those of the facilitators.

5 Integrating the Three RE Practices

In an attempt to mitigate the problems faced with the use of the three practices, we developed an approach for integrating these practices. In this section, we present the lessons we have learned from integrating the three practices. First, three success factors of the integration are presented. Second, we introduce the setup workshop as the means to perform the actual integration.

5.1 Success Factors of the Integration

Teamwork. We identified teamwork as the first success factor of the integration. By definition, teamwork produces synergy as people with different skills work together towards a common goal. Team members create shared meanings and arrange this knowledge into common frames. We found out that teamwork was essential for the identification of: 1) the project-specific needs of the requirements specification; 2) the project-specific needs for co-operation and communication; 3) the key information streams (informants and targets) in different stages of the project, and 4) the types of practices that are beneficial for the project. As a consequence of teamwork, the decisions were based on an extensive knowledge base and debate, they were far more willingly accepted by group members, and the reasons behind them were commonly understood.

Correct timing of the RE practices. The second success factor of the integration is the correct timing of the RE practices, so that each practice is used at the right phase of the project – maybe a seemingly trivial task, but not so. As an example, one specialist commented that the utilization of the specification template too early restricts the creative work and collaboration too much. The selection of the proper sequence of the practices also involves planning when and how the different stakeholders are to be engaged in specifying requirements. Without this kind of planning, the first involvement of some key stakeholders was often in peer reviews, late in the phase of specifying requirements, causing frustration on the part of the stakeholders and extra development costs.

Tailoring the RE practices. Tailoring the selected practices was crucial, as none of them was found to be optimal as such. Here, tailoring means a purposeful activity in which a practice or practices are adapted to fit a particular purpose. We found out that two types of tailoring were used. First, the case study organizations performed

company-level tailoring of the practices. The most typical type of company-level tailoring was to define the structures and scrutinize examples of different types of specification templates by adapting a standard version like the IEEE Recommended Practice for Software Requirements Specifications [6]. Second, project-specific tailoring was needed.

Purposeful, project-specific tailoring seemed to require at least one skilled person to take ownership of the tailoring. In particular, tailoring the three practices as integrated required an understanding of their overlapping characteristics. Increased knowledge about the variety of practices and their strengths and weaknesses seemed to help the team to choose the appropriate types of practices and adapt them to their context. When software project teams participated in collaborative workshops to a greater degree, specification templates were less necessary for guiding the writing work and peer reviews for information sharing.

As a result of the tailoring, the variability of the collaborative workshops increased. The goals and required participants of the collaborative workshops varied. Hence, it was considered crucial to choose and communicate the type, procedure, and goal of each collaborative workshop in order to avoid unnecessary participants and false expectations.

5.2 Setup Workshop as a Crucial Component of the Integration

A setup workshop is a collaborative and facilitated workshop used for planning and communicating how to utilize the three practices in the creation of requirements specification in software projects. The setup workshop was originally developed and piloted in Company A, and later refined and piloted in Company B. The setup workshop seems to be a key component for integrating the three practices. The current version of the setup workshop is presented in the following and illustrated in Figure 2.

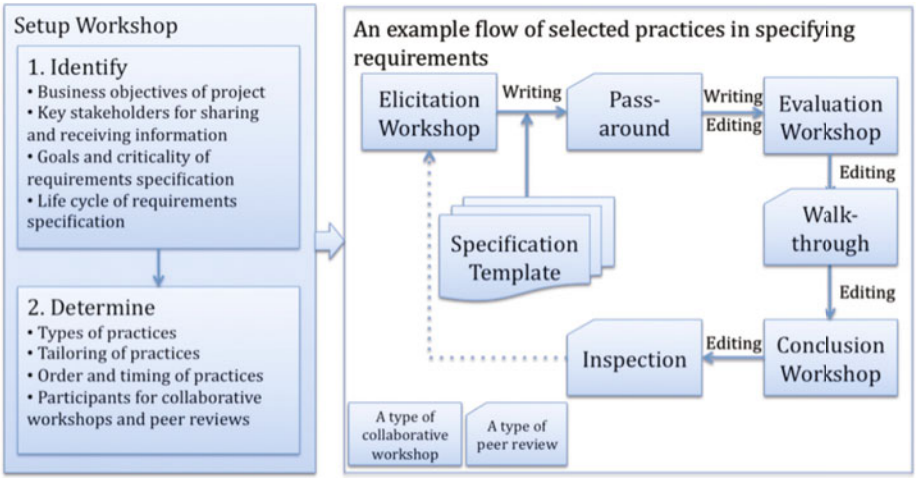


Fig. 2. Using a setup workshop for planning the integrated deployment of the three RE practices

The setup workshop is performed in the early stages of specifying requirements. In the setup workshop, the facilitator enables the participants to identify or clarify the following:

- the business objectives of software project,
- the key and minor informants of requirements in different stages of the project,
- the project-specific goals of the requirements specification and the needs of its readers,
- the criticality of the requirements specification and the role of oral communication, and
- the life-cycle of the requirements specification and the estimated volatility of requirements.

Immediately after identifying the above factors, the facilitator supports the participants in determining the types of practices, the ways they are tailored, and their sequence. At the same time, they determine the participants for the planned collaborative workshops and peer reviews.

The benefits of this approach were obvious. First, the readers of the requirements specification considered that this approach improved the relevance of specifications. Second, from setup workshop participants' point of view, proper planning and communication of the goals of each collaborative workshop and peer review was considered to reduce false expectations and frustration on the part of participants and to promote early information sharing between stakeholders. Overall, 20 out of 28 practitioners who participate in specifying requirements for information systems in Company B perceived the idea of the setup workshop as very useful for their work. The remaining eight practitioners perceived it as useful. Remarkably, all three team leaders and seven out of eight project managers found the setup workshop very useful.

A setup workshop was found useful, at least, in software projects where a requirements specification had an important role. An informant commented that the use of the three practices in an intertwined way was also found beneficial in certain software projects in Company B, even without applying the setup workshop as such. This implies that the integration of the practices can be successfully performed in several ways and the setup workshop is just one approach to perform it.

During our study, Company A was transformed from a traditional software organization to an agile one, which changed the company's software development culture. Moreover, the software developers increasingly opposed documentation and peer reviews. Consequently, the use of requirements specification was reduced and peer reviews nearly discontinued. As a result of these changes, we were not able to fully apply setup workshops as initially piloted; oral communication took the key role as the main information channel. Interestingly, software developers appeared to apply collaborative workshops informally. Even though the applicability of setup workshops as such can be limited in agile software organizations, we, in fact, suggest their use as a tool to help agile software development teams towards efficient information sharing between cross-functional teams.

6 Discussion and Conclusions

Although specification templates, collaborative workshops, and peer reviews have all been recommended, they are typically treated as independent RE practices in the literature. Our findings indicate that the independence of the practices leads to several problems in practice. The use of specification templates often leads to individual specifying work, resulting in relatively long documents. Peer reviews are typically performed too late and reviewers are not motivated to contribute.

The integration of the three practices was identified as a rational way to reduce such problems. As each of the three practices has strengths and weaknesses that partly overlap, using the practices in a tailored and intertwined way helps the team to reduce the negative influences of their weaknesses. As a means to perform integration, our findings suggest using the setup workshop for planning the workflow of specifying requirements, identifying project-specific ways to collaborate, and selecting and tailoring the appropriate types of practices. The use of a setup workshop can improve the applicability of specification templates and promote early information sharing between stakeholders. Proper planning and communication of the goals of each collaborative workshop and peer review should reduce false expectations and frustration on the part of participants.

The proposed integration approach includes limitations and challenges when adopting it in a software organization. Software developers may not easily adopt the proposed integration approach, if they already oppose meetings, documentation, and peer reviews. In addition, the tailoring of the RE practices, as a key element of the integration, requires more RE skills than the use of standardized practices. A software organization needs to consider whether they have the necessary skills or willingness to acquire them. Furthermore, the integration approach will increase the variability of the used RE practices in the software organization. The work practices and requirement specifications in different software projects will become less comparable. Consequently, a process owner and management may find it more difficult to observe the progress and quality of software projects.

The significance of our findings is to be confirmed in future studies. The role of certain authors as active participants in the companies may have affected the construct validity of the results. It should be noted that the development of the integration approach partly occurred as everyday work in two companies and was not solely organized to support the research purpose. Furthermore, we were able to apply the integration approach only in two out of the five companies. This weakens the external validity of the findings. New studies in several organizations are needed to evaluate whether the integration approach really addresses to the problems faced with use of the three RE practices.

While this paper presented the three RE practices as integrated, in future, it would also be worth studying how to integrate larger sets of RE practices that are adopted for use in software companies. Indeed, evaluating the usefulness of the setup workshop for planning and tailoring the entire RE process of a software project appears to be a promising idea.

References

1. Hofmann, H.F., Lehner, F.: Requirements Engineering as a Success Factor in Software Projects. IEEE Software 18(4), 58–66 (2001)
2. Katasonov, A., Sakkinen, M.: Requirements Quality Control: a Unifying Framework. Requirements Engineering 11(1), 42–57 (2006)
3. Tsumaki, T., Tamai, T.: Framework for Matching Requirements Elicitation Techniques to Project Characteristics. Software Process Improvement and Practice 11(5), 505–519 (2006)
4. Fitzgerald, B., Russo, N.L., O'Kane, T.: Software Development Method Tailoring at Motorola. Communications of the ACM 46(4), 64–70 (2003)
5. Uusitalo, E.J., Komssi, M., Kauppinen, M., Davis, A.M.: Linking Requirements and Testing in Practice. In: Proceedings of the 16th IEEE International Requirements Engineering Conference, pp. 265–270. IEEE CS Press, Barcelona (2008)
6. IEEE Recommended Practice for Software Requirements Specifications (IEEE Std-830), pp. 207–244 (1998)
7. Robertson, S., Robertson, J.: Mastering the Requirements Process, 2nd edn. Addison-Wesley, Boston (2006)
8. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. Wiley, Chichester (1997)
9. Davis, A.M.: Just Enough Requirements Management: Where Software Development Meets Marketing. Dorset House Publishing, New York (2005)
10. Wiegers, K.E.: Software Requirements, 2nd edn. Microsoft Press, Redmond (2003)
11. Brockmann, R.J.: Where Has the Template Tradition in Computer Documentation Led Us? In: Proceedings of the 2nd Annual International Conference on Systems Documentation, pp. 16–18. ACM, Seattle (1983)
12. Kaner, C., Bach, J., Pettichord, B.: Lessons Learned from Software Testing–A Context-driven Approach. Wiley, New York (2002)
13. Power, N., Moynihan, T.: A Theory of Requirements Documentation Situated in Practice? In: Proceedings of the 21st Annual International Conference on Documentation, pp. 86–92. ACM, San Francisco (2003)
14. Gottesdiener, E.: Requirements by Collaboration: Getting It Right the First Time. IEEE Software 20(2), 52–55 (2003)
15. Kylmäkoski, R.: Efficient Authoring of Software Documentation Using RaPiD7. In: Proceedings of the 25th International Conference on Software Engineering, pp. 255–261. IEEE Computer Society Press, Portland (2003)
16. Carmel, E., Whitaker, R.D., George, J.F.: PD and Joint Application Design: a Transatlantic Comparison. Communications of the ACM 36(6), 40–48 (1993)
17. Purvis, R., Sambamurthy, V.: An Examination of Designer and User Perceptions of JAD and the Traditional IS Design Methodology. Information & Management 32(3), 123–135 (1997)
18. Wood, J., Silver, D.: Joint Application Development, 2nd edn. Wiley, New York (1995)
19. Davidson, E.J.: Joint Application Design (JAD) in Practice: The Journal of Systems and Software 45(3), 215–223 (1999)
20. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Upper Saddle River (2001)
21. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison-Wesley, Boston (2003)
22. Wiegers, K.E.: Peer Reviews in Software: A Practical Guide. Addison-Wesley, Massachusetts (2001)

23. Wiegers, K.: When Two Eyes Aren't Enough. Software Development 9(10), 58–61 (2001)
24. Avison, D., Lau, F., Myers, M.D., Nielsen, P.A.: Action Research. Communications of the ACM 42(1), 94–97 (1999)
25. Avison, D.E., Baskerville, R., Myers, M.D.: Controlling Action Research Projects. Information Technology & People 14(1), 28–45 (2001)
26. Potts, C.: Software-Engineering Research Revisited. IEEE Software 10(5), 19–28 (1993)
27. Coghlan, D.: Insider Action Research Projects: Implications for Practising Managers. Management Learning 32(49), 49–60 (2001)
28. Gummesson, E.: Qualitative Methods in Management Research, 2nd edn. Sage Publications Inc., Thousand Oaks (2000)
29. Yin, R.K.: Case Study Research – Design and Methods, 3rd edn. Sage Publications Inc., Thousand Oaks (2003)

A Case Study on Tool-Supported Multi-level Requirements Management in Complex Product Families

Margot Bittner, Mark-Oliver Reiser, and Matthias Weber

Technische Universität Berlin
Fakultät IV - Softwaretechnik, Sekr. TEL 12-3
Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
{margot,moreiser,we}@cs.tu-berlin.de

Abstract. [**Context & Motivation**] Despite numerous advancements in product family engineering over the past decade, the management of highly complex product families still remains a significant challenge. In our previous work, we presented the multi-level approach for pragmatically planning and managing variability and reuse across independent product ranges, thus avoiding the unmanageable complexity of a rigid product line infrastructure on the global level above these individual product ranges. [**Question/problem**] The multi-level approach has not yet been extensively validated in industry in the area of requirements management of product families. [**Principal ideas/results**] A major tool-supported case study from the automotive domain in the area of body comfort electronics was set up and performed in order to validate the multi-level approach for requirements management in complex product families. [**Contribution**] The results from the industrial case study demonstrate the applicability of the multi-level approach and emphasize its benefits in an industrial setting. Furthermore, important lessons were learned leading to numerous refinements and extensions both in concept and tool support.

1 Introduction

Product-line oriented development is one of the new paradigms of software development proposed over the past decades [1,2]. According to this paradigm, the focus of development is shifted from individual software products to the overall set of products a software manufacturer has on offer, i.e. his *product line* or *product family*. Instead of developing the products in parallel and independently from one another, only a single, but variable product—called the *product line infrastructure*—is built; the actual products offered to the customer are then derived from that infrastructure through configuration. A key objective of all product line approaches is to make the product line itself a genuine, tangible entity within the development and evolution process and to strategically manage the commonality and variability between individual product instances within the product line's scope.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 173–187, 2010.

This basic idea of product line orientation well suits the situation in automotive industry, with its huge product ranges and its extensive variability. The same applies to many other industrial domains of software-intensive systems. However, when applying traditional product line methods and techniques to a highly complex product family, such as that of a global automotive manufacturer, the engineer is faced with a dilemma: managing everything as a single, gargantuan product line is virtually impossible owing to its enormous complexity; but when dividing the range of available products into several smaller independent product lines, systematic reuse and strategic variability management across these portions—two of the key benefits of product line orientation—are lost. It is the purpose of the multi-level approach, as presented in [3,4], to avoid this strict alternative by offering a compromise between a single global and several smaller, independent product lines. With this technique, it is possible to split up a huge product line into smaller, independent *sublines* but still, to strategically steer their commonalities and variabilities on a global level.

In this paper we present the results of an automotive case study of tool-supported multi-level requirements management and we will discuss the experiences and the lessons learned from it and show how they motivated refinements and extensions to the existing approach and tool. After a brief overview of the multi-level approach in the coming section and a short description of the applied tool-support in Section 3, we describe the background, scope and the quantitative results of the case study in Section 4. In the main part of this article we will then describe in detail the experiences from the case study and the resulting refinements and extensions to the approach and its tool-support (Section 5). The last section finishes with a summary and several concluding remarks.

2 The Multi-level Approach

The basic intention of the multi-level approach is to allow for strategic planning and to manage development across several smaller, independent product lines, without introducing a large, rigid product line infrastructure on the global level. To achieve this, we turn to the development artifacts of two or more independent, lower-level product lines and initially assume that these artifacts are defined and evolved independently for each lower-level product line. To now allow for a coordination on the global level, we introduce an *additional* artifact of the same type which has the sole purpose of making proposals for the content of the lower-level artifacts, thus serving as a template for them; the individual proposals in this template may or may not be adopted within the lower-level artifacts. The template artifact on the global level is called a *reference artifact*, whereas the lower-level artifacts are called *referring artifacts*; whenever a lower-level artifact diverges from a proposal in its reference artifact, we speak of a *deviation*.

In addition to this, the proposals in the reference artifact are marked as optional or obligatory, which allows one to recognise deviations in the referring artifacts as *legitimate deviations*, that constitute a deviation from an optional proposal, or *illegitimate deviations*, that deviate from an obligatory proposal. If

a referring artifact contains no deviations at all or only legitimate deviations, then we say that this artifact *conforms to its reference artifact*. This is also called the *conformance state* of a referring artifact.

With this mechanism at hand, we can now use the reference artifact to steer the contents of the referring artifacts: we can start by making optional proposals in order to give the development teams working on the referring artifacts a chance to leisurely tailor their local content towards the global template; later, we can turn these, step by step, into obligatory proposals in order to enforce a tighter alignment to this global vision. All along the way, we can track the current progress of individual lower-level product lines by investigating the deviations within the corresponding referring artifact.

This general idea of a multi-level management of artifacts can be applied, in principle, to any type and form of artifact. For the purpose of this article, however, we concentrate on experiences from a case study involving requirements specifications.

A detailed description of the technical realization of this conception cannot be given here. Instead, we provide a rough overview as a basis for the discussions in the following sections. Each artifact may point to another artifact as its reference artifact and is then called a referring artifact. Each element within a referring artifact may point to an element of the reference artifact as its reference element, and is then called a referring element. We thus have two types of so-called *reference links*[1]: the first, from a referring artifact to its reference artifact and, the second, from a referring element to its reference element. With these reference links we can find out which element of the reference artifact serves as a template for a particular element of the referring artifact. In addition, all conceivable forms of deviation within the lower-level artifact are identified and so-called *deviation permissions*, which can be used to mark the content of the reference artifact as optional or obligatory by allowing or disallowing certain forms of deviation, are provided.

The possible forms of deviation as well as the semantics of the deviation permissions were defined formally [4] and can therefore be used to automatically derive logical constraints on the referring artifacts, which need to be met by each referring artifact in order to conform to its reference artifact. These logical constraints can then be checked automatically to highlight legitimate and illegitimate deviations and to determine the overall conformance state of a referring artifact. All this was presented in detail in [4], so more information on the technical realization of the multi-level approach can be found there.

3 Tool Support Used in the Case Study

The generic multi-level approach to development artifacts in general, as outlined in the previous section, was, in a second step, specifically tailored to requirements

[1] The notion of reference links should not be confused with the notion of configuration links [5]; the latter serve to establish a logical relation between the configurations of two or feature models.

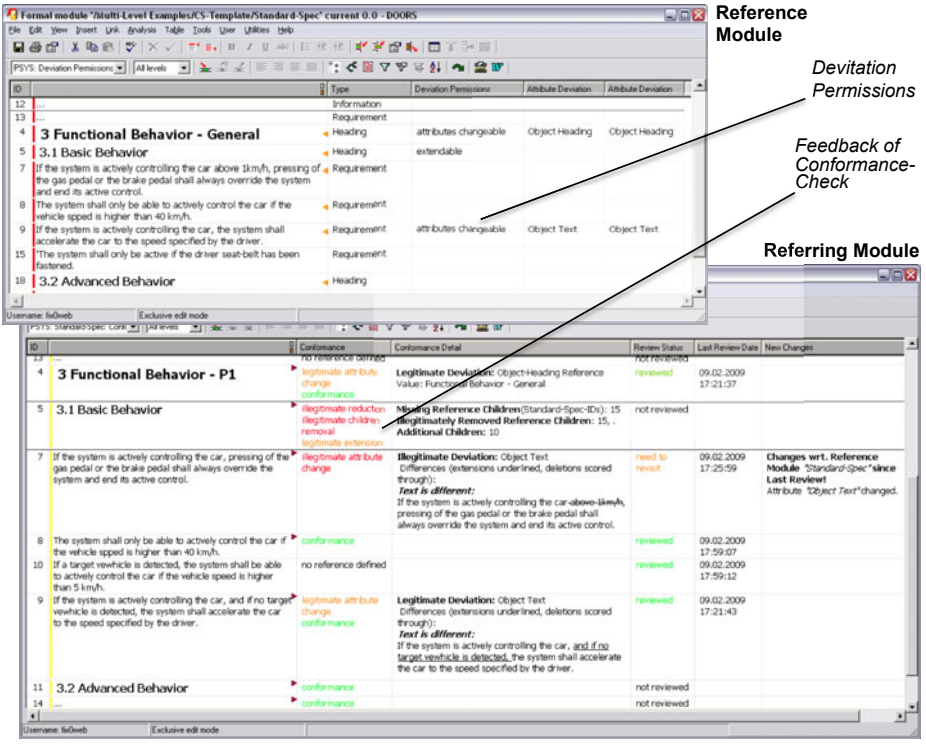


Fig. 1. The tool for multi-level DOORS modules. The lower screen shot shows the feedback of the conformance-check algorithm for each object: its conformance, differences to its reference object (if any), review status, and recent changes.

specifications, defined and managed within the commercial tool Rational DOORS. To denote this special instantiation of the multi-level approach we speak of *multi-level* DOORS *modules*, instead of multi-level requirements artifacts, following the terminology in DOORS where a single requirement specification container is called a "module". The tool support for this specialized multi-level technique was implemented as an extension to DOORS. It has been applied in the N-Lighten case study as mentioned above.

The reason for building on an existing commercial tool was that this tool is widely used in the automotive industry and it provides a very flexible extension mechanism in the form of the DOORS Extension Language (DXL), which actually constitutes a complete programming language.

A screen shot of DOORS running the tool for multi-level requirements management is shown in Figure 1. The upper window shows a reference module which is referred to by the module in the lower window, the specification presented in the upper window thus serves as a template for the one in the lower window. As can be seen in the upper window, the deviation permission attributes all go in one single column, i.e. a single DOORS custom attribute was created for them. The

attribute lists the values of all deviation permission attributes that differ from the default value. In the lower window, on the other hand, it can be seen that the feedback of the algorithm checking the conformance of the referring module is also presented in a dedicated column/attribute, called `Conformance`. The conformance state for each object can be found here. Conformance violations are highlighted by a special background color.

In other implementations of the multi-level approach mentioned in [4], the reference link from a referring model to a reference model is defined inside the referring model. In contrast, in the implementation presented here the links referring to reference modules are kept in one or several separate DOORS modules, called *synchronization module(s)*, solely dedicated to the purpose of defining what modules are used as reference modules and what other modules are referring modules that have to conform to them. This way, it is not necessary to put special objects into referring modules.

Further details on the tool will be given in Section 5 where the extensions resulting from the case study will be described.

4 The N-Lighten Case Study

The case study mentioned at the beginning of this article, the so-called *N-Lighten case study*, was intended for further evaluation and refinement of the multi-level approach both on a conceptual level as well as on the level of its supporting tool implementation. The study was specifically focused on requirements specifications maintained in the tool Rational DOORS. In this section, we first give an account of this case study's background, content and objectives before going into the details of the lessons learned in Section 5.

The N-Lighten case study was based on several technical specifications of an automotive body electronics system. The lights and illumination function group, was chosen as the subject of investigation, hence the name of the case study. It covers such functionalities as blinker lights, interior lights and coming-home lights. The involved specifications contained between 1.300 and 3.500 requirements, which represents a scale sufficient for evaluating the concepts under realistic conditions. The actual editing of the requirements during the case study was conducted by persons who were not involved in the conceptual development of the multi-level approach and were therefore initially unfamiliar with the concepts and tool. This was a valuable challenge in order to assess the approach's understandability and its overall feasibility under realistic conditions.

At the beginning of the survey, three existing technical specifications were used. These were available as three DOORS modules: one for the lights and illumination functionality of a car platform A, one for the same functionality of a car platform B, and a third which originally served as the template of the other two, i.e. from which the other two modules were once derived by way of copy and paste (cf. Figure 2). In the following, these three DOORS modules will be referred to as platform A module, platform B module and base module respectively.

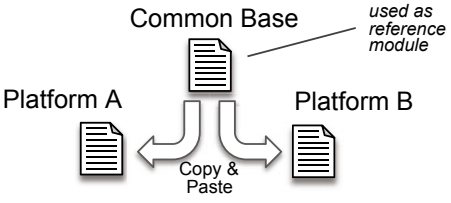


Fig. 2. The three DOORS modules involved in the case study

This initial setting perfectly matches the intention of subscoping and the multi-level approach: the overall product line, comprised of platform A *and* platform B cars, is split up in two sublines, namely platform A and platform B, and each subline is enhanced independently; the multi-level management now allows the commonalities and differences between these two sublines to be tracked and strategically coordinated without the necessity of introducing a rigid product line organization. As one of the most important intentions and benefits of the multi-level approach, the variability *between* platform A and platform B no longer appears as variation points within the platform A and platform B specifications, which substantially reduces the complexity of these individual specifications.

In order to set up a multi-level hierarchy with all three specifications, an auxiliary synchronization module (as described in Section 3) which provided the tool prototype with the necessary meta-information had to be created. This is very straightforward and mainly defines the hierarchy of reference and referring DOORS modules involved, in this case the base module as a single reference module and the platform A and platform B modules as referring modules. Then, the reference links between the requirements in the platform A and platform B modules and their corresponding reference requirements in the base module had to be established and defined in DOORS in a form suitable for the tool prototype. This was a more daunting task as is discussed below. Together, these preparations allowed the two referring specifications to be managed according to the multi-level approach, for example to define deviation permissions in the base module, to reveal illegitimate deviations in the platform A and platform B modules or to propagate changes, e.g. a newly added object, from one of the referring modules to the base module.

The platform A module contained 1.351 objects whereas the platform B module contained 3.501. This alone shows the remarkable difference in system complexity between a low-end and a medium-class vehicle, not to mention luxury-class models. Table 1 presents some statistics of the case study that further characterizes the specification modules involved. Out of such a statistic, several interesting facts become immediately obvious—an important benefit of the multi-level concept. For example, the number of non-referring objects, i.e. objects without a reference object, adequately measures how much additional information is introduced in a subline, compared to the base module. In order

to provide such information in a normalized form, we introduce two statistic measures: coverage and innovation.

Definition. Given a reference artifact R with n_R elements and a referring artifact A with n_A elements of which n_{ref} have a reference link defined, the value cov_A^R defined as

$$cov_A^R = \frac{n_{ref}}{n_R}$$

is called *R-coverage* of A.

A referring artifact's coverage is an adequate measure of how much information from the base module found its way into the referring module: a value of 1 signifies that all information from the superline is somehow represented in the subline artifact (but it may have been extensively changed, as indicated by the number of deviations) whereas a value of 0 indicates that no information from the superline is left in the subline[2].

In addition to an artifact's coverage, it is possible formulate a measure for the artifact's inventiveness:

Definition. Given a reference artifact R and a referring artifact A with n_A elements of which n_{ref} have a reference link defined, the value $innov_A^R$ defined as

$$innov_A^R = \frac{n_A - n_{ref}}{n_A}$$

is called *R-innovation* of A.

An artifact's innovation reflects how much additional information was introduced in a subline, in comparision with the information taken from the superline. A value of 1 means that the subline artifact consists purely of newly introduced information, whereas a value of 0 shows that no information was newly introduced in the artifact.

Another interesting fact to note about the two specifications, is that the structuring of the original base module was conversely changed in the two subline specifications: in the platform A module the hierarchy was flattened and in the platform B module it was deepened. This caused a conspicuous difference in the ratio of refinements to reductions in the two subline modules. From a case study perspective, this was an interesting additional test for the multi-level concept; in practice however, it will usually only occur in early phases of adopting the concepts, where the structure of the base module is still fairly unconsolidated.

[2] Please note that this definition assumes that the advanced concepts of split and merge, as introduced later in this article, are not allowed; however, even if split and merge occurs in an artifact this definition is usually sufficiently accurate.

Table 1. Statistics for the platform A and platform B specifications

	DOORS Modules		
	Base	Platform A	Platform B
Objects, thereof ...	1,908	1,351	3,501
- without reference object		590 43.7%	1,714 49.0%
- with reference object		761 56.3%	1,787 51.0%
Coverage		0.399 39.9%	0.937 93.7%
Innovation		0.437 43.7%	0.490 49.0%
Deviations:			
- Refinement		33 2.4%	107 3.1%
- Reduction		24 1.8%	18 0.5%
- Move		5 0.4%	36 1.0%
- Reorder		–	2 0.1%
- Textual Changes		220 16.3%	241 6.9%
- Merge		–	2 0.1%
- Split		65 4.8%	16 0.5%

5 Lessons Learned and Extensions

Based on the results of the N-Lighten case study, it was possible to draw several interesting conclusions, which in turn led to a number of significant extensions to the multi-level concept. These will be detailed in this section. Fortunately, we only extended and did not have to change the technique, i.e. the basic idea of the multi-level approach remained unaltered, and therefore the earlier publications on the subject remain valid without modifications.

Creating Reference Links. As briefly indicated above, one of the necessary activities for setting up the multi-level hierarchy proved to be a rather tedious task: the creation of reference links on the object level, i.e. from the objects in the platform A and platform B modules to their corresponding reference object in the base module. The most difficult part is deciding which object from the base module is the correct reference object for a certain lower-level object, or if no such object exists. Except for the cases where an equal object is found in the reference module, this requires that the detailed semantics of the two specifications are taken into consideration. Given the large number of 1.351 to 3.501 objects in the two lower level specifications, this difficulty is of high practical relevance.

However, for several reasons this difficulty does not invalidate the basic idea and concept of the multi-level approach:

1. The problem described here only occurs if a multi-level hierarchy is set up on the basis of preexisting legacy specifications. If the platform A and platform B modules were initially created with the multi-level tool support in mind, then the reference links would have been established automatically when copying the base module, i.e. when creating a new subline.
2. But even in the case of legacy specifications, the problem can be alleviated by providing appropriate tool support. A tool could pre-define the reference

links based on an appropriate similarity measure and assist the user in reviewing and adapting them by way of an appropriate on-screen presentation. While in the case study some ad-hoc scripting was used to mechanize some of the link creation, the design of a more sophisticated *linking assistant* would be relatively straightforward; experience in the model transformation field where a similar issue exists, could be applied. It is realistic to assume that the linking problem could be reduced quite substantially in this way.

3. However, even without such a linking assistant, the task of manually establishing a substantial number of reference links is absolutely manageable, as the case study clearly showed. And this was the case, even though the persons defining the reference links did not know the content of the three specifications before starting work on this case study.

Therefore this difficulty does not actually represent a critical problem for this approach.

Set of Reference Modules. In our earlier publications, it was assumed that each referring artifact may only have a single reference artifact (or in the context of DOORS each referring DOORS module may only have a single reference module). This way, the explanations and examples could be kept as straightforward as possible. However, it is perfectly viable to also allow more than one reference artifact/module for a single referring artifact/module. This additional flexibility does not mean any significant changes for the basic concept of multi-level management because the detailed definition of deviations and deviation permissions does not rely on the reference link of the entire artifact, but on the reference links of individual objects within the artifact, and these may still only point to a single reference object each (at least for now; that restriction will be discussed in the next paragraph). This additional flexibility merely provides the possibility of merging the objects from several modules on the reference level in a single referring module, i.e. to package the elements differently into models. For example, it is possible to take one subtree of the containment hierarchy in a referring DOORS module from reference module model A and another subtree from another reference module B.

Split and Merge. One of the most important observations from the case study was that a splitting and merging of objects in a referring module is of great practical relevance, leading to referring objects with more than one reference object and vice versa. Let us first investigate the precise meaning of split an merge in our context. When a DOORS object has more than one reference object, this means that two separate objects from the reference level were semantically merged into a single object in the referring module. Since this in itself always represents a change in the lower-level artifact with respect to the reference artifact, we can perceive this as an new form of deviation called "merge":

> **Definition.** When a referring element has more than one reference element, the referring element is assumed to comprise the semantic meaning of all reference elements. This form of deviation is called a *merge*.

Similarly, several referring objects may point to one and the same object as their reference object. The usual practical motivation for this is that a single object's semantic meaning is distributed within the referring module among several distinct objects. Again, this necessarily constitutes a deviation in itself:

> **Definition.** When several referring elements have the same reference element, the referring elements are assumed to jointly comprise the semantic meaning of the reference element. This form of deviation is called a *split*.

These two novel forms of deviation now complement the types of deviation formally defined in Section 4.3 of [4].

Having precisely defined split and merge, we can now turn to the technical details of the concept. Most importantly, the fact that splitting and merging is now allowed has an impact on the precise definition of the other deviations. For example, if an object o with name "abc" has two reference objects o_A with name "abc" and o_B with name "xyz", how do we decide if a change in o's name has occurred? As a general rule, we use a logical disjunction of the old definition evaluated separately for each referring object (in case of a split) or each reference object (in case of a merge). In the given example this means that we assume that o's name has changed if its name is different from o_A's name *or* its name is different from o_B's name. In the above example this would be true. With this general semi-formal rule, the logical constraints from our earlier publications can be straightforwardly applied to split and merge as well (cf. Table 2 in [4]).

Standard Permissions and Vetoes. In the early, prototypical version of the tool, it was necessary to define the deviation permission for each object separately. In order to reduce effort and to simplify maintenance of the deviation permissions, but also to allow enforcement of a strict process of handling multi-level systems, the tool now provides a mechanism to define standard permissions for a DOORS module which are then assigned to all objects contained in the module.

Changes Within a Subline Module. The multi-level concept is useful for identifying and managing changes in a referring module with respect to the reference module. However, during the case study a similar question often occurred: what changes were introduced in a referring module, e.g. the platform A module, with respect to an earlier version of this same referring (platform A) module. After some consideration, this issue was deemed outside of the scope of the multi-level concept. It can and should be solved orthogonally to the multi-level concept, plainly on the level of a change management facility. While this observation did not lead to an extension of the multi-level approach, it was still beneficial in further clarifying the distinction between the multi-level approach and traditional change management.

Review Status. This was introduced to make the multi-level approach manageable for very large referring modules, which are typically reviewed or reworked

over such long a period of time (e.g. weeks or even months) that new changes in the reference module occur during the review. To provide support for such use cases, each reference relation has two status attributes attached: `ReviewStatus` and `NewChanges`. The first attribute has the default value "not reviewed" and can be manually set to "reviewed", for example after a review of the conformance analysis and a possible adaptation of the synchronization settings. The date of this setting is displayed in the attribute (last review date). If the corresponding reference object has been changed since the date of the review, then (after synchronization) the attribute `ReviewStatus` is set to "need to revisit", and the attribute `NewChanges` contains a delta description of what has been changed. As previously, the user can then inspect these changes and reset the review status to "reviewed" once more.

Bidirectional Synchronization. The multi-level approach is mainly intended as a means of strategically steering the content of the referring modules by providing voluntary or obligatory templates within the reference module(s). To realize this technically, the deviation permissions (attached to the reference module's objects) were defined formally and can thus be used to automatically derive logical constraints on the referring modules which need to be met by these referring modules in order to conform to the reference module (cf. Section 2; all detailed in [3,4]). The automatic checking of these logical constraints, called *conformance checking*, was the main functionality already provided by early, prototypical versions of the tool for multi-level requirements management. The editing of the reference and referring modules, however, was initially seen as a purely manual process.

However, experience from the case study showed that in addition to the conformance check algorithm another functionality is often desired: to automatically resolve conformance violations and to propagate newly introduced changes from the referring module to the reference module or vice versa. This can be seen as a two-way synchronization mechanism with which changes on referring level or reference level can be propagated up or down, respectively. For example, when new requirements for some innovative car functionality are being added to the platform B module (i.e. the referring module for the platform B vehicle range), these additional requirements first appear as a deviation of the platform B module and can then be automatically propagated to the reference level if they are to become a (voluntary or obligatory) template for the platform A model as well; from this moment on, the lack of this new requirement will show up as a (legitimate or illegitimate) deviation in the platform A model.

This two-way synchronization was realized in the tool by way of so-called *synchronization actions*, i.e. instructions telling the synchronization algorithm if and how a certain deviation is to be propagated up or down. These actions are specified by the user in an additional, dedicated DOORS attribute within the source module. In particular, the following propagation (or synchronization) actions are provided: restore attributes, restore node (i.e. restore an object which was deleted), restore position, restore order, remove extensions, restore reductions, remove out-link extensions, and restore out-link reductions. As can be seen, for each of the forms of deviations identified for DOORS modules and for each of

the deviation permission attributes, a corresponding synchronization action is provided.

Furthermore, these synchronization actions can be set automatically based on the result of a pervious conformance analysis. Two different schemes of automatically setting synchronization actions are available: (1) "restore conformance" sets all synchronization actions so as to ensure that all referring objects become fully conforming, i.e. all remaining deviations become legitimate; (2) the scheme "overwrite all" erases any deviation in the referring module, effectively resetting its content to that of the reference module (obviously, this scheme should be used with due care).

Finally, the same mechanisms that have been introduced for defining standard deviation permissions, c.f. "Standard Permissions and Vetoes" above, have also been introduced for synchronization actions. Taken together, these various mechanisms allow for very powerful analysis and synchronization settings with a wide range of practical applications (e.g. document generation, requirements exchange between manufacturer and supplier) which we are just beginning to explore.

In addition to these rather conceptual observations, there are also several more technical aspects to note:

Comparison of OLE Objects. In order to uncover deviations in a subline module, it is of paramount importance to compare two attribute values and, if they are unequal, to show what has been changed. DOORS provides a special functionality for editing attribute values which proved to be a daunting challenge in this regard: the value of a text attribute may contain OLE objects. OLE stands for "Object Linking and Embedding" and is the name of a distributed object system and protocol on the Microsoft Windows platform [6], which provides a mechanism to embed editable objects of various types and origins within the document of a third-party tool. For example, an image drawn in Microsoft PowerPoint or a table created in Microsoft Excel can easily be included in the text of a textual attribute in DOORS.

The problem is that a DOORS DXL script cannot easily compare two attribute values for equality which contain such OLE objects, because this would mean to comparing the content of the OLE objects, which in turn would require full support of the objects' internal format, for example the data format of an MS Excel sheet. Since OLE objects can have an arbitrary data format, it is not possible to find a principle solution here, and even implementing a comparision for only the most important types of OLE objects would be an enormous effort. Furthermore, OLE objects in DOORS are used extensively in practice and avoiding them is absolutely unrealistic.

Fortunately, a solution was found at least for comparing two OLE objects for exact equality. This is sufficient for checking whether any change occurred, but not for finding out or presenting to the user, what was changed or whether the change was only superficial and should rather be ignored, e.g. a differing column width in a table. Given the fact that the OLE objects used in DOORS requirement specifications are usually not too complex—due to the imperative of requirement

atomization [7]—this partial solution proved sufficient for practical use. However, in full-fledged tool support, a more sophisticated comparison mechanism for the most common types of OLE object, probably diagrams and tables imported from Microsoft Word or Excel, is desirable.

Visibility of Reference Links. Since the tool DOORS does support filtering on the objects of a requirements container but not on the (incoming or outgoing) links displayed in a module, it becomes confusing for the user to see which links leaving or entering his module are reference links connected to the multi-level system and which links are outside this system. While this is a general DOORS issue, we found that the Multi-Level approach should at least not aggravate it. Therefore, the multi-level tool support allows the user to "hide" all reference links in an additional special attribute and only display them as required.

Conformance and Review Summary. Initially, the tool was designed only to present the conformance state inside the referring module (as a dedicated DOORS attribute of each referring object). While this is the most obvious place for it and suitable in many use cases, it has turned out that it is often very convenient for attaining an overview of all conformance states in all referring modules at a glance. Therefore, a functionality was implemented in the tool to show a special view summarizing the conformance states in all its referring modules: for each object in the reference module this view displays the conformance state within the reference module and the review status for its referring object in each referring module, each in a separate column (cf. Figure 3).

In summary, the following general conclusions could be drawn regarding the feasibility of the overall multi-level approach:

Forms of Deviation are Appropriate. During the conceptual design of the multi-level approach, several different forms of deviation were identified and corresponding deviation permissions had to be attached to the tree structure of DOORS objects. A multitude of different distinctions and constructions would have been conceivable here; the aim was to find a solution which is as simple as possible and still sufficiently differentiated to allow the formulation of all distinctions required in practice.

The case study showed that the forms of deviation and how they cover changes of the hierarchical structure of the specification artifacts matches the deviations we came across in the three investigated authentic specifications very well. The only important exception is the merging and splitting of objects, but appropriate forms of deviation were defined and added to the approach, as was discussed above.

Reference Links are Real-World Entities. Semantically relating the objects of the subline modules (i.e. platform A and platform B modules) to objects in the base module through reference links, did not pose any substantial problems from a conceptual point of view. It might be a lot of work to initially define those

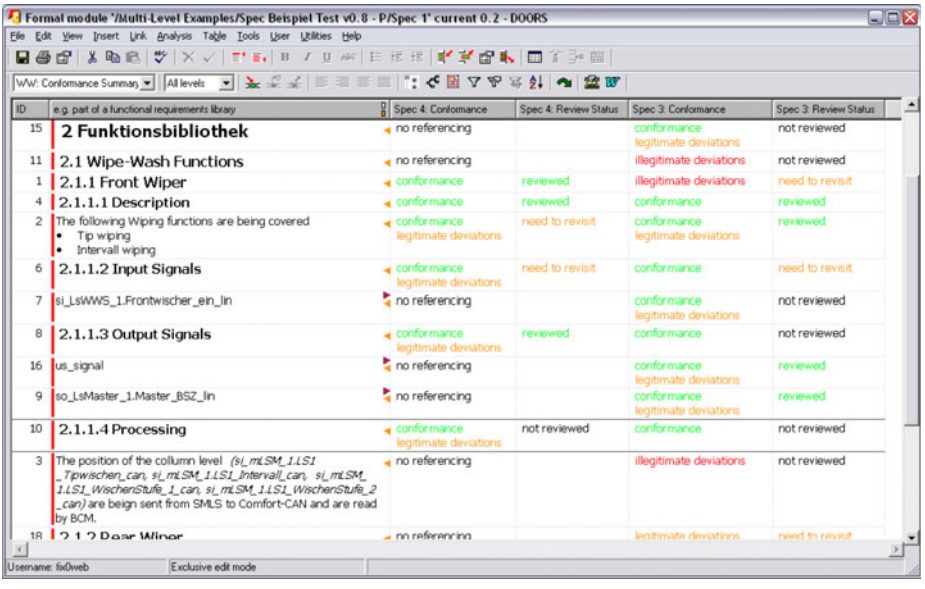


Fig. 3. A reference module showing a conformance summary for several referring modules

links (as discussed above), but for any two particular objects it can usually be decided easily and definitely if one of the two should serve as the reference object of the other. This was the case, even though the structure of the base module had been changed substantially in the subline modules; even if an object was put in a different section, it was still possible to clearly decide which reference object it should receive.

This suggests that a reference link is not an artificial concept, introduced solely to make the multi-level approach work, but rather it captures an actual conceptual relation which naturally occurs in real-word use cases.

Adequate Usability. The overall usability of the tool for multi-level DOORS modules proved to be quite reasonable. Even though it was not specifically designed for usability in the first place, it was still quite convenient for performing the typical actions, such as spotting deviations, editing deviation permissions, finding out whether deviations are illegitimate, or propagating deviations up and down.

Considering these observations and the overall size of the specifications—over three thousand objects in the case of the platform B module—we are confident that together with the improved tool support the multi-level approach is now immediately applicable in practice.

6 Conclusion

This article presented the so-called N-Lighten case study which was conducted on industrial specifications and by engineers who were not involved in the

development of the multi-level approach. This means that the approach had to be taught to them beforehand, putting the understandability and feasibility of the concepts as well as the related tool support to the test.

Initially, the tool described in Section 3 was mainly intended as a research prototype for experimentation purposes. Thanks to the above study, some internal case studies and the resulting refinements and extensions, part of which were described in Section 5, we are convinced that both the concepts as well as the tool support are now ready for application in industrial development projects. The tool is publicly accessable on a website, together with documentation and a tutorial [8].

Acknowledgments. The research leading to these results has received funding from the European Community's 7th Framework Programme under grant agreement no. 224442. In addition, the authors would like to thank Martin Becker for his commitment and valuable feedback.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Reading (2002)
2. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
3. Reiser, M.-O., Weber, M.: Managing highly complex product families with multi-level feature trees. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), pp. 146–155. IEEE Computer Society, Los Alamitos (2006)
4. Reiser, M.-O., Weber, M.: Multi-level feature trees – a pragmatic approach to managing highly complex product families. Requirements Engineering 12(2), 57–75 (2007)
5. Reiser, M.-O.: Core concepts of the Compositional Variability Management framework (CVM). Technische Universität Berlin, Technical Report, no. 2009-16 (2009)
6. Chappell, D.: Understanding ActiveX and OLE. Strategic Technology Series. Microsoft Press, Redmond (August 1996)
7. Pohl, K.: Requirements Engineering — Grundlagen, Prinzipien, Techniken. Dpunkt Verlag (2007)
8. DOORS Multi-Level Tool Web-Site (2010), `www.mule-re.org`

A Domain Ontology Building Process for Guiding Requirements Elicitation

Inah Omoronyia[1], Guttorm Sindre[1], Tor Stålhane[1],
Stefan Biffl[2], Thomas Moser[2], and Wikan Sunindyo[2]

[1] Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
{inah.omoronyia,guttorm.sindre,tor.stalhane}@idi.ntnu.no
[2] Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
{stefan.biffl,thomas.moser,wikan}@tuwien.ac.at

Abstract. [Context and motivation] In Requirements Management, ontologies are used to reconcile gaps in the knowledge and common understanding among stakeholders during requirement elicitation, and therefore significantly improve the quality of the elicited requirements. **[Question/problem]** However, a precondition of state-of-the-art ontology approaches for requirements elicitation is an existing domain ontology. While this is not a trivial precondition, there are only a few reports on approaches to systematically and efficiently build domain ontologies, and these approaches are often highly biased towards their intended use. **[Principal ideas/results]** In this paper, we investigate an approach for building domain ontologies suitable for guiding requirements elicitation. We evaluate the feasibility of the approach based on a real-world industrial use case by analyzing natural language text from technical standards. **[Contribution]** A major outcome is that the proposed approach can help reduce the effort of building domain ontologies from the scratch.

Keywords: Requirements elicitation, domain ontology, semantic analysis, natural language processing, domain engineering.

1 Introduction

The RE process [17] starts out with a specification which is informal, opaque, and dominated by personal views, while the goal is to have a specification, which is formal, complete, and reflects the stakeholders' common view. The use of an ontology can help to tackle these challenges: from opaque to complete because an ontology can encode knowledge about the domain, thus ensuring that important requirements are not forgotten, and from personal to common view because an ontology defines a standard terminology for the domain, which mitigates misunderstandings about terms. If the ontology is defined in a formal language, it will also help regarding the formality

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 188–202, 2010.

dimension. There has been an increasing interest in using ontologies to aid the RE process.

Ontologies are specifications of a conceptualization [4] in a certain domain. An ontology seeks to represent basic primitives for modeling a domain of knowledge or discourse. These primitives are typically concepts, attributes, and relations among concept instances. The represented primitives also include information about their meaning and constraints on their logically consistent application [5]. A domain ontology for guiding requirements elicitation depicts the representation of knowledge that spans the interactions between environmental and software concepts. It can be seen as a model of the environment, assumptions, and collaborating agents, within which a specified system is expected to work. From a requirements elicitation viewpoint, domain ontologies are used to guide the analyst on domain concepts that are appropriate for stating system requirements.

There are a number of research approaches to elicit and analyze domain requirements based on existing domain ontologies. For example, Lee and Zhao [13] used a *domain ontology and requirements meta-model* to elicit and define textual requirements. Shibaoka et al. [18] proposed GOORE, an approach to goal-oriented and ontology-driven requirements elicitation. GOORE represents the knowledge of a specific domain as an ontology and uses this ontology for goal-oriented requirements analysis [12]. A shortcoming of these approaches is the need for a pre-existing ontology, as to our knowledge there is no suitable method for building this ontology for requirements elicitation in the first place in an at least semi-automated way. In industrial settings, the task of building domain ontologies from scratch can be daunting, mostly due to the size of technical standard documents that need to be interpreted by domain experts and the wide range of domain concepts coverage that will be the input to such ontologies. Therefore, the domain ontology building task can greatly be leveraged by tool support.

This paper explores the challenge in building a domain ontology that is sufficient for guided requirements elicitation. Firstly, we investigate an approach for building domain ontologies from existing technical standards which the specified requirements need to be compliant with. Our investigation is based on a set of heuristics used for extracting semantic graphs from textual technical standards to generate compatible baseline domain ontologies. Secondly, we present an evaluation of the feasibility of our approach and provide insights on the challenges of semi-automatically building domain ontologies using natural language texts. The remainder of this paper is structured as follows: section 2 presents related work and motivates the research issues; section 3 discusses the characteristics of a suitable ontology for requirements elicitation and also proposes an approach for achieving such ontologies. Section 4 presents the evaluation of our approach and a discussion of lessons learned during this research. Finally, section 5 concludes the paper and presents some ideas for further work.

2 Related Work and Research Issues

Natural Language Processing (NLP) techniques are important when analyzing text to extract domain ontologies for requirements elicitation. NLP generally refers to a

range of theoretically motivated and computational techniques for analyzing and representing naturally occurring texts. The core purpose of NLP techniques is to achieve human-like language processing for a range of tasks or applications [15]. The core NLP models used in this research are part-of-speech (POS) tagging and sentence parsers. POS tagging involves marking up the words in a text as corresponding to a particular part of speech, based on its definition, as well as its context. On the other hand, sentence parsers transform text into a data structure (also called parse tree). Such data structure provides insight into the grammatical structure and implied hierarchy of the input text [1]. Standford parser/tagger[1] and OpenNLP[2] are the core set of NLP tools used in this research. Tag meanings used are from the Penn Treebank project, which involved the annotation of a corpus consisting of over 4.5 million words of English. Words were annotated for part-of-speech (POS) information and skeletal parse structure [16].

Research on domain engineering is also critical to understand an approach to analyze text with the aim of extracting an ontology for requirements elicitation. Domain engineering highlights the process of reusing domain knowledge in the production of new software systems. Domain engineering particularly aims to support systematic reuse, focusing on modeling common knowledge in a problem domain [2]. Sowa's work on conceptual structures [19] introduces a synthesis of logic, linguistics, and Artificial Intelligence as a mechanism for domain knowledge representation.

A closely related research contribution regarding textual extraction of domain ontologies from natural language style requirement documents is the case study on the application of natural language processing to domain modeling presented by Kof [10]. Kof views the domain ontology itself as a valuable and reusable requirements engineering product. He presented three steps for the extraction of a domain ontology which include: *term extraction*, *term clustering and taxonomy building* as well as *finding associations between extracted terms*. The domain model to be extracted is built using the extracted terms as well as the associations between them. Kof [12] also proposed an approach using NLP techniques to construct an initial system model by extracting knowledge from existing requirement texts. Kof [9, 11] furthermore presents mechanisms for analyzing textual scenarios using computational linguistics. The outcome of this analysis process is the identification of whether communicating objects or whole actions are missing in a text. The viewpoint of computational linguistics here is inclusive of NLP and theories about the linguistic knowledge that humans need for generating and understanding written language. Flores [3] has also explored an approach that uses semantic filtering techniques for the analysis of textual requirements descriptions. Flores postulated that filtering relevant text fragments according to semantic criteria enhances large textual requirements description processing. In addition, Flores proposed the use of a linguistic technique known as the *Contextual Exploration Method*, to extract semantically relevant sentences in order to support requirements analysis and validation. Four semantic viewpoints were considered and included: *concepts relationships*, *aspecto-temporal organisation*, *control* and *causality* statements.

[1] http://nlp.stanford.edu/software/lex-parser.shtml
[2] http://opennlp.sourceforge.net/

Although Kof's and Flores' methods are based on analysing natural language texts to extract an a ontology that can subsequently be used for requirement elicitation, no analysis has been done on the suitability or usefullness of the resulting ontology for such purpose. The association of concepts in a domain ontology can be described by its taxonomy or by the use of axioms. The taxonomy is a hierarchical system of concepts, while axioms are rules, principles, or constraints guarding the relations amongst concepts. Furthermore, the level of granularity to which the axioms are specified is highly influenced by its intended use within the ontology [6]. From the viewpoint of using domain ontologies for requirements elicitation, axioms specify the extent to which such an ontology can be useful for the categories of questions to which the ontology can provide answers.

Again, existing related works lack insight into the potential challenges of extracting a domain ontology from a textual source. For instance, such text might not sufficiently describe the domain of concern or contain terms that are not unique to the domain being described. Furthermore, derived ontology from analyzed text can contain unique concepts/relations in the domain of interest which do not contribute to the requirements elicitation process. In such a case, the text analyzed contains valid domain terms that do not necessarily contribute to useful domain ontology. In this research, we reckon that each of these challenges needs to be investigated as to how it can be mitigated.

To address the research issues identified in closely related work, this paper discusses the semantic features of suitable domain ontologies for requirements elicitation and proposes a process for systematic and efficient domain ontology building. We then evaluate the feasibility of our approach based on a real-world industrial use case by analyzing text from technical standards.

3 Ontology Suitable for Guided Requirements Elicitation

For a domain ontology suitable for requirements elicitation, its competence as determined by its axioms is vital. This is particularly true when investigating the use of such ontologies for tracing high-level goals to concrete requirement representations as well as for obtaining insight into the quality of the written requirements. Axioms specified on a relation between two concepts in an ontology aim at providing more meaning to the relation or involving concepts. Richer relational axioms thus suggest more insight on written requirements that reflects corresponding concepts. In this section we highlight three semantic features for enriching the axioms for requirements elicitation ontology.

- *Explicit relational expression*: In addition to the inherent properties of relations such as *transitivity*, *symmetry* and *sub-classes*, requirements elicitation ontologies also aim at associating specific semantic attributes that have domain specific implications. This approach is in contrast for instance to the work of Kitamura et al. [8], who used static predefined stereotypes in naming relations. For example, the relation between the two concepts agent and message using stereotypes is represented by *agent<requires>message*. The predefined stereotype *<requires>* hides the semantic implication of the nature of the relation (e.g., the relation could imply send, receives, blocks etc.). Such otherwise hidden semantics could be useful in

guiding the analyst in determining the relevance of a prescribe trace inference from the ontology to the system being specified.

- *Qualified identification of relations*: Ontologies used to support computers in reasoning will normally identify relations by the use of a single so-called interesting or *performative* verb. These are verbs whose action is accomplished merely by saying them. Performative *verbs* such as *requires, sends,* or *request*, explicitly convey the kind of act being performed by a concept by virtue of an involving relation. But considering an ontology for guided requirements elicitation, the semantic implications of such performative verbs are normally described in an adjoining qualifier such as adjectives and conjunctions. Thus, it is more insightful to name the relation between agent and message using the identifier "periodically sends" rather than only using "send". In this research we explore the use of performative verbs in combination with their qualifiers to semantically identify relations between concepts.
- *Temporal and spatial expressions*: For using domain ontologies for requirements elicitation, we need insight into temporal and spatial implications of relations that exist between concepts. For example, assume A, B and C are concepts in a domain and the description "A requires B during C" is a feature used to characterize a domain. For semantic insight during requirements elicitation, it is important that the explicit relation that exists between A and B as well as the temporal relation that A has with C are captured by the ontology and made obvious to the analyst.

Building domain ontologies with the above semantic features is challenging as it requires domain experts to describe and document their knowledge about the domain with the meaning of concepts and implied relations in a detailed manner, which will be time-consuming. In this research, we explore a rule-based approach that uses NLP techniques to evaluate the possibility of automatically capturing initial or baseline domain ontology from existing text.

3.1 Rule-Based Baseline Ontology Extraction

The basis of this approach is: given some pre-processed textual document and some predefined heuristics based on NLP, it is possible to extract ontology concepts and relations that are semantically meaningful for requirements elicitation. The pre-processing of the document is normally a manual process and ensures that the text from which concepts and relations are to be extracted is suitable for sentence-based analysis. This includes the removal of symbols or formatting from text that will otherwise alter the meaning of extracted concepts or relations. It is worth mentioning that the more detailed a document is pre-processed, the more effective domain ontologies can be extracted from the natural language text. In contrast, for large documents such detailed preprocessing is difficult, as it requires more effort from the domain experts. This challenge for large documents necessitates an (semi-)automated pre-processing approach to help improve the resulting ontology. In the first instance, the rule-based ontology extraction investigates two automated document pre-processing mechanisms known as *bracket trailing* and *bridged-term completion*. Subsequently, Subject-Predicate-Object extraction, association mining and concept clustering is executed on the pre-processed text.

Bracket trailing: Textual descriptions normally use bracket pairs or dashes as punctuation marks to set apart or interject supplementary text within other texts. A common use for brackets in the writing of technical standards is to indicate a reference within the text. They are, however, also frequently used to provide explanatory words or phrases. It is most common for the bracketed text to be used within a single sentence and these texts can be seen as pointers to the concepts represented in a particular sentence [14]. Given the sentence: *"A PLC with a safe transmission protocol (see figure x) shall be restricted to the communication end devices (F-Host, F-CPU, F-Device and F-I/O-Module)"*, the aim of bracket trailing is to filter out pre-determined reference pointers such as "figure x" from existing brackets. Subject/object concepts are then extracted from the remaining text within the bracket. Extracted subjects/objects are finally related to the *head subject* or *object* depending on whether the bracket is used within the noun phrase (NP) or verb phrase (VP) of the sentence. Relations derived via bracket trailing are semantically identified using the stereotype <refers to>. For the example, the concepts "F-Host", "F-CPU", "F-Device" and "F-I/O-Module" are extracted by bracket trailing and related to the concept "devices".

Bridged-term completion: Given the phrase "an input or output device is required by the system", NLP analysis will generate "input", "output device" and "system" as potential concepts. For this example, an understanding of the context of the phrase, suggests that the combined terms "input device" rather than just "input" more completely describe the concept in an unambiguous way. Such ambiguity in concept identification is common in text and it is normally left to the reader of the text to decipher their contextual implications. In this research, an ambiguous term is referred to as bridged-term, indicating that a human interpretation is required to capture its semantic implication. Bridge-term completion involves a semi-automated process of discovering and correcting bridged-terms in textual documents using observed patterns in a sentence parse tree. The occurrence of a learned pattern in an analyzed text raises a flag to the domain expert highlighting possible concept ambiguity and a potential more complete set of terms that better describes the concept.

Figure 1 shows the NP tree patterns in which bridged-terms can occur. The leaf nodes in case 1 consist of a singular noun node or a sequence of singular nouns (NN) nodes separated with commas (,) on the left of *and/or* conjunction. Left of the *and/or* conjunction are a sequence of initial NN nodes where the last node is a NN, plural noun (NNS), proper singular noun (NNP) or proper plural noun (NNPS). On the right hand side of the *and/or* conjunction there are no commas. The NP pattern labeled case 2 is similar to case 1. The core distinction is that in case 2 the first leaf node to the left of *and/or* conjunction can also be a JJ qualifier and there is no comma separating the first two leaf nodes in the sequence. During Bridged-term completion, the NP pattern described in figure 1 is parsed, and a recommended set of more complete terms to describe the subject/object concept in the NP tree is deduced based on node combination (see shaded section of figure 1). The example above corresponds to case 1 and yields "input device" and "output device" as potential subject concepts. A flag indicating possible concept ambiguity with the recommended set of more complete terms describing the concept is then brought to the notification of the domain expert.

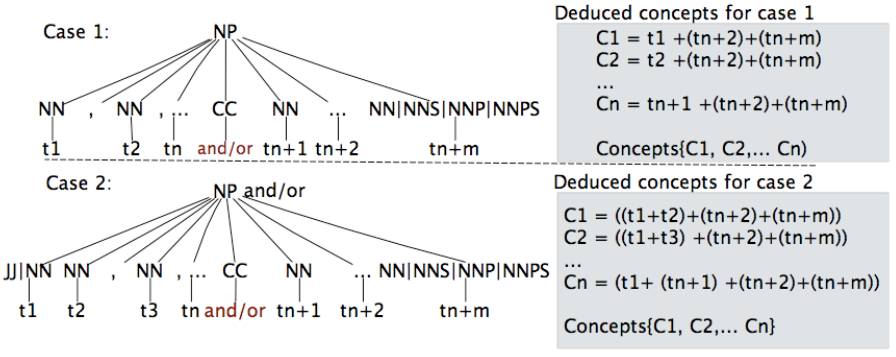


Fig. 1. NP parse tree pattern for bridged-terms

Subject-Predicate-Object (SPO) extraction: SPO is a parsing process that involves navigating the phrase tree of a sentence to extract declarative clauses with their predicate, associated subject and object. Each sentence clause can be said to consist of a noun phrase (NP) and a verb phrase (VP). The NP contains the subject, which can be identified by a noun POS variant (e.g., singular, plural or proper noun) as leaf nodes and can further be semantically qualified by adjectives (JJ) and associated quantities (CD). The VP contains the predicate and the object. The identified subjects and objects are both concepts in the baseline ontology, while the predicate defines the semantics of the relation between the concepts.

When parsing the VP to extract predicate relation, verb variant leaf nodes (e.g., past tense (VBP), present tense (VBZ), etc.) and their first sibling proceeding/preceding qualifiers (e.g. JJ, adverb (RB) and preposition (IN)) are extracted. Similarly, a NP or VP is parsed to extracted noun variation, qualifier and quantity. Extracted nodes are concatenated to a string representation of either the potential predicate relation, subject or object concept. Finally, subject/object concepts extracted during SPO analysis are characterized as head/derived concepts during *association mining*. The *head subjects and objects* act as the domain and range of a relation identified by extracted predicates.

Association mining: Association mining identifies head and derived subjects/objects from the set generated during subject/object extraction. It also extracts other types of relations that are not captured during the predicate extraction process. While the relations between subjects and objects in the predicate extraction are explicitly defined based on the terms that exist in the VP, the relations extracted during association mining are inferred based on any prepositional phrase (PP) whose first preceding sibling node is a noun phrase.

Prepositional phrases normally consist of a preposition and an object of a preposition. Terms that exist in the potential subject/object set and also act as an object in the propositional phrase, are considered as derived subjects/objects. The subjects/objects contained in the first preceding NP are considered as the head subjects/objects. Objects/subjects in a sentence without PP are by default head objects/subjects. From a semantic viewpoint, a preposition is used to illustrate temporal or spatial relationship

between the objects/subjects of the prepositional phrase and objects/subjects of the preceding sibling NP. Such inferred relational semantics are identified and represented by the stereotype <temporally infers> and <spatially infers> respectively. Consider the example above "*A PLC with a safe transmission protocol…*" where the concept "*PLC*" is the head subject while "*safe transmission protocol*" us a derived subject. The preposition "*with*" suggests a spatial relation between the two identified concepts.

Concept clustering: SPO analysis and association mining is likely to result in replication of concepts and relations across sentences. During concept clustering, lexical similarity matching is used to firstly merge the different semantic graphs to eliminate repetitive concepts and relations, and secondly to generate a taxonomy tree based on similarity between terms used to represent different concepts. Vector Space Model techniques have been investigated as the lexical similarity matching approach.

Overall, the rule-based baseline ontology for requirements elicitation involves the following steps: (1) Manual textual document pre-processing to ensure that the text being analyzed is suitable for sentence-based analysis; (2) Automatic bracket trailing filters out predefined reference pointers and extracting relevant concepts referring to a head concept within the sentence; (3) Semi-automated bridged-terms updating to remove ambiguous concepts; (4) Automatic sentence analysis that extracts subjects/objects and predicates as concepts and relation amongst concepts; (5) Automatic association mining to extract temporal and spatial references amongst concepts; and (6) Automatic concept clustering to build a taxonomy of concept.

4 Evaluation and Discussion

This section discusses an initial empirical study of the proposed approach for extracting domain ontologies suitable for requirements elicitation. By using real-world industrial technical standards – in our case from the domains of transport, Adaptive Cruise Control (ACC) - we compare the domain ontology generated by our rule-based approach with a manually generated domain ontology to understand the implications of our approach. We focus on the challenge of irrelevant terms that are not unique to the domain being described and thus do not contribute to the requirements elicitation process. We also evaluate if the different extracted concepts and relations from the analyst, domain expert or via rule-based approach were semantically intuitive and meaningful for guided requirements elicitation. Finally, we present lessons learned.

4.1 Manually Generated and Rule-Based Comparison of Elicitation Ontology

The manual generation of the requirements elicitation ontology involved two experienced participants. The first, who acted as the analyst, had only a vague understanding of ACC but were knowledgeable about requirements elicitation processes, while the second participant had a much deeper insight into the ACC domain and acted as the domain expert. Both participants were knowledgeable of how concepts and relations amongst concepts can be used to generate an ontology.

Paragraphs from two representative sections in ISO 15622 – ACC systems technical standard [7] were presented to both participants (see figure 2 labeled case 1 and 2).

Our selection criterion was a document section that was representative and at the same time provided insights independent of the initial manual pre-processing. This is because the level of manual document pre-processing carried out by a third-party on the selected text can influence the quality of domain ontologies generated using the rule-based approach. Both participants were asked to extract concepts and generate relations among the concepts from the text. To understand the implications of our approach, we feed the same natural language texts into an implemented prototype for automated rule-based ontology approach.

Case 1
The main system function of Adaptive Cruise Control is to control vehicle speed adaptively to a forward vehicle by using information about: (1) ranging to forward vehicles, (2) the motion of the subject (ACC equipped) vehicle and (3) driver commands (see Figure x). Based upon the information acquired, the controller (identified as "ACC control strategy" in figure x) sends commands to actuators for carrying out its longitudinal control strategy and it also sends status information to the driver.

Case 2
Type 1a and 2a ACC systems shall either temporarily suspend operations but remain in the ACC-active or transition to ACC-stand-by if the driver depresses the clutch pedal. For type 2a systems, the Automatic brake maneuver can be continued during the use of the clutch pedal. After the system releases the brakes, the system may either resume ACC control or transition to ACC-stand-by state in response to the driver depressing the clutch.

Fig. 2. Sample text presented to participants (Source: ISO 15622 [7])

Table 1. Number of concepts and relations extracted from sample text

	Concepts		Relations		
	Assumed	Explicit	Assumed	Explicit	Parent-Child
Rule-based	7	30	16	15	21
Analyst	0	26	0	25	0
Domain expert	3	18	8	13	6

Table 1 shows the number of concepts and relations extracted from the sample text. Explicit concepts/relations are directly inferred from the text, while assumed concepts/relations are inferred using reasoned based on concept clustering for the rule-based approach or based on understanding and knowledge of the analyst respectively the domain expert. For each of the categories, a higher number of concepts and relations were identified using the rule-based approach compared to those identified by the domain expert or analyst. Insight from participants showed that since the analyst had a limited understanding of the domain, concept/relation extraction was strictly based on his/her understanding of the sample text. On the other hand, the domain expert relied more on his/her general understanding of the domain to assimilate the meaning and implication of each concept/relation from the sample text. Both participants used one hour to analyze and document a domain ontology based on the sample text. The automated rule-based approach used all the required steps besides the initial manual document pre-processing step. Based on the above sample text, this result is an initial indicator that the automated rule based approach can help reduce the effort in generating requirements elicitation ontology and at the same time achieve a greater coverage of domain concepts. On the other hand, it is also important to understand if

the additional concepts and relations extracted by the rule-based approach are valid, semantically meaningful and necessary and not simply an *over specification*.

Using the ontology extracted by the rule-based approach and manually by the analyst and domain expert, this study focuses on getting insights into the over specification (extracting concepts and relations that are not necessary) or under specification (missing concepts and relations that are otherwise necessary) using our rule-based approach; and if the extracted concepts and relations were semantically intuitive for guided requirements elicitation. We discuss each of these factors and pinpoint how they can be possibly ameliorated.

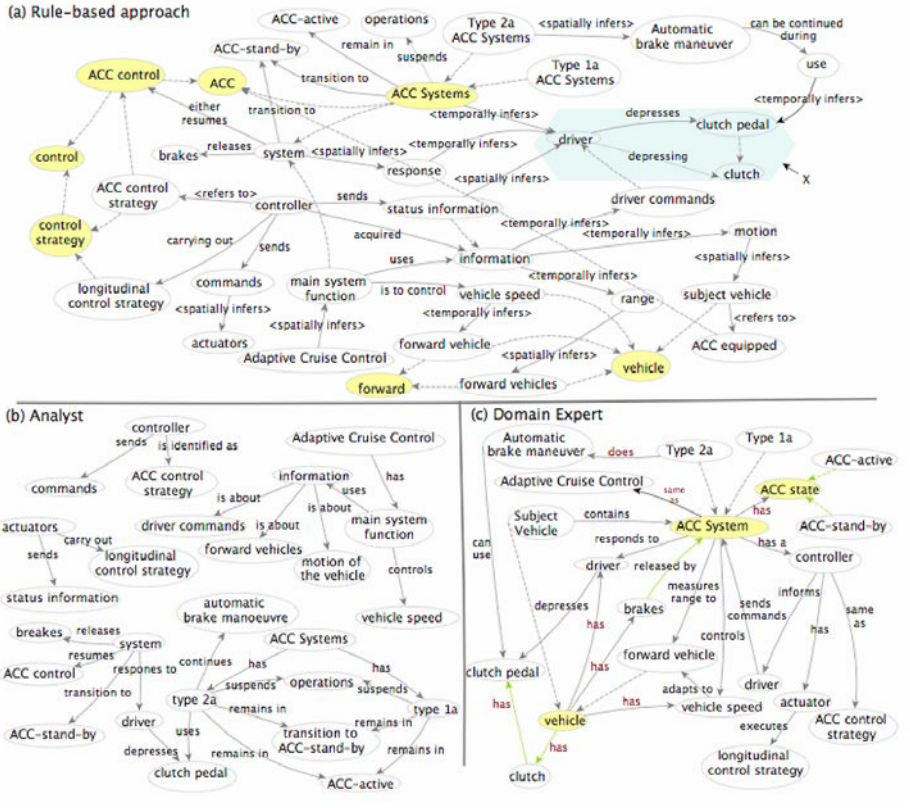


Fig. 3. Rule-based and manually generated ontology

Ontology over specification: It is difficult to state if an ontology is over specified since there is normally no initial understanding of the concepts and relations contained in the domain ontology to be used for a specific requirements elicitation task. Rather, the purpose of this study is to get an understanding of how over specification of concepts and relations can occur for a requirements elicitation domain ontology based on our rule-based semantic analysis of natural language text. Rule-based analysis of the section of sample text "*…adaptively to a forward vehicle by using*

information about: (1) ranging to forward vehicles…" (see figure 2 case 1) results in the over specification of the concepts "*forward vehicle*" and "*forward vehicles*". Apart from the understanding that one concept is singular while the other is plural, the two concepts point to the same semantic meaning. Thus, not much additional insight is obtained by modeling the two as separate concepts. A case of relational over specification using the rule-based approach is demonstrated in the modeling of the relation between "*driver*", "*clutch pedal*" and "*clutch*" concepts (see the shaded section of figure 3a and marked X). In this case, only the relation "*depresses*" between "*driver*" and "*clutch pedal*" is meaningful, while the relation "*depressing*" between "*driver*" and "*clutch*" is not required, given that the former relation implies the later one.

The two cases of over specification pointed out above can be eliminated using stemming/lemmatizing of concept terms to their root words or a more rigorous concept and relational clustering methods. In the first case, the term "*vehicles*" can be stemmed to its root form "*vehicle*", while in the second case, the relation "*depresses*" and "*depressing*" can be merged to a single relation between "*driver*" and "*clutch pedal*" or "*clutch*". On the other hand, modeling of generic terms such as "*control*", "*use*" and "*forward*" in the rule-based approach as concepts can be considered an over specification for domain ontologies suitable for requirements elicitation. Such generic terms are more suitably defined in general ontologies such as Wordnet [3]and can hence be filtered out from specific domain ontologies.

Ontology under specification: Initial insight into under specification when using the rule-based approach can be carried out by checking if all semantically meaningful concepts and relations captured by the analyst and domain expert can be directly or indirectly inferred from the domain ontology generated by the rule-based approach. 24 of the 26 concepts captured by the analyst were also capture by the rule-based approach. As further discussed below, the two remaining concepts "*motion of vehicle*" and "*transition to ACC-stand-by*" is considered wrongfully modeled or less meaningful. 20 of the 21 concepts captured by the domain expert were also captured by the rule-based approach. The remaining concept "*ACC state*" is the concept captured by domain expert which was not represented in the rule-based approach. A follow-up of this finding from the domain expert suggested that "*ACC state*" was informed by his/her understanding that although ACC-stand-by and ACC-active were only mentioned in the text, the two concepts were the possible states of ACC system. Hence the concept "*ACC state*" captured as the parent of ACC-stand-by and ACC-active by the domain expert. On the other hand, it had only been possible for the rule based approach to conceptualize "*ACC state*" if the sample document (case 2 figure 2) was rewritten as "*… but remain in the ACC-active state or transition to ACC-stand-by state…*". In this case, both "*ACC-stand-by state*" and "*ACC-active state*" would be identified and represented by the rule-based approach as concepts while ACC state would be represented as a parent concept.

Three relations present in the ontology created by the domain expert were not captured by the rule-based approach. These include: "*vehicle*" has "*driver*", "*vehicle*" has "*brakes*" and "*ACC system*" same as "*Adaptive Cruise Control*". In all three cases, the sample text was not sufficient and did not contain possible references to suggest

[3] http://wordnet.princeton.edu/

such relation among concepts and was hence impossible to infer for an automated approach. Overall, insights from this study demonstrate that the challenge of under specification using rule-based approach can be reduced by either rigorous manual preprocessing of text document; providing sufficient text for rule-base analysis or by domain experts manually adding the missing concepts and relations.

Semantically intuitive and meaningful ontology: For the ontology generated by the analyst, the concept "motion of vehicle" is wrongly modeled. The reason for this is that in the sample text, the concept "*information*" is precisely related to "*the motion of the subject vehicle*" and not to every "*vehicle*". Similarly, the concept "*transition to ACC-stand-by*" confounds the already modeled relation between "*transition to*" that exist between the concepts "*system*" and "*ACC-stand-by*". The concepts "*actuators*" and "*longitudinal control strategy*" are similarly modeled using the relation "*carry out*" and "*carrying out*" for the domain ontology generated by the analyst and by the rule-based approach respectively. However, from a linguistic viewpoint, the expression "*actuators carry out longitudinal control strategy*" is a complete self-defining phrase, while the expression "*actuators carrying out longitudinal control strategy*" suggests the need for an additional support phrase. In this example, the defined relation between "*actuators*" and "*longitudinal control strategy*" from the analyst is semantically more intuitive than the relation generated by the rule based approach. Such linguistic issues in the definition of relations for the rule-based approach can possibly be reduced if during the predicate extraction phase of the SPO analysis, the root form of the verb gerund or present participle (VBG) is used.

The outcome of the study also suggests that domain ontologies originating from the rule-based approach and from the domain expert should complement each other. This is because concepts and relations are sometime better modeled using the rule-based approach than the domain ontology created by the domain expert, and vice versa. A core observation of the comparison of the domain ontology created by the rule-based approach and the one created by the domain expert is that relations between concepts can sometimes be represented in a rather concise but semantically equivalent and meaningful way. For example, as shown in figure 3a, the relation between the concepts "*Automatic brake maneuver*" and "*clutch pedal*" is identified using an intermediate concept "*use*" with <temporally infers> and "*can be continued during*" relational identifiers between them. In the domain ontology created by the domain expert (figure 3c), a more concise relational identifier "*can use*" links the two concepts "*Automatic brake maneuver*" and "*clutch pedal*". On the other hand, the rule-base approach also demonstrates cases where the use of concepts as intermediaries provides more insights into the relational semantics. For instance, the ontology created by the domain expert (figure 3c) relates the two concepts "*controller*" and "*driver*" via the relation "*informs*". While this is a semantically valid relation, the token that is transmitted from controller to driver is not an explicit characteristic of the relation. Rather, in the rule-based ontology, the "*controller*" and "*driver*" concepts are related via an intermediate concept "*status information*" with <spatially infers> and a "*sends*" relational identifier between them. In this case, the conceptualization of "*status information*" provides more details on the token that is transmitted from the controller to driver.

4.2 Lessons Learned and Limitations of Rule-Based Approach

The core lesson learned in this research is that domain ontologies for supporting requirements elicitation can be achieved by extracting knowledge from technical documents. The domain ontology manually generated by an analyst has shown to be more prone to error when identifying concepts and relations than the ontology that is automatically generated. This is understandable, since analysts usually have no knowledge of the ontology domain. The ontologies created by the rule-based approach and by the domain expert can be used to complement each other. Thus, a viable technique for building requirements elicitation domain ontologies is to generate a baseline ontology using the rule-based approach based on the technical documents and then let it be verified and refined by domain experts.

Furthermore, manual document pre-processing before carrying out sentence based NLP analysis that extracts concepts and relations is critical but in non-trivial cases difficult to achieve. This is because the generated ontology is highly dependent on the quality and format of source text. Our general experience is that domain standard texts tend to conform to good linguistic style and in some cases use controlled language subsets. This is normally not the case when source of the text is informal documents such as emails, interview transcripts and web pages. The successful application of the rule-based ontology generation approach has so far been validated for a domain standard text, and hence might not be a valid approach for informal text sources. Automated document pre-processing such as bracket trailing and bridged-term completion, where possibly ambiguous terms are brought to the notice of the domain expert, are viable options to reducing manual preprocessing effort.

Bridged-terms completion can sometimes raise false alerts. For instance, the sentence "*Safety communication and standard communication shall be independent*" will alert the domain expert on possible concept term ambiguity, even though "*safety communication* and "*standard communication*" are both completely defined concepts. As part of our future work, we plan to investigate a machine learning approach to reduce such false positives. Bracket trailing relies on the assumption that it is common for the supplementary material in a bracketed text to provide more information on the particular single sentence. Such an assumption cannot hold for writing styles where a bracketed text is used to provide supplementary material that references multiple sentences.

As in most text analysis techniques, a 100% precision/recall is difficult to achieve although a high precision/recall rate for the rule-based approach can be inferred for the text used for the initial study. In the first case, a high precision is inferred based on the analysis of sample text for over specification. Two cases of concept over specification were captured out of 37 concepts (95% concept precision). Similarly, two relations were over specified out of 51 relations (96% relational precision). In the second case, a high recall is inferred based on the analysis of sample text for under specification. The analysis showed that 20 of the 21 concepts captured by the domain expert were also captured by the rule-based approach (95% concept recall). Similarly, three relations present in the domain expert ontology were not captured by the rule-based approach (94% relational recall). Given that this is an initial preliminary study using a relatively small subset of technical standard text, more studies will be required to generalize this outcome for a much larger subset.

This preliminary study reveals a scalability concern. Using the rule-based approach, a small snippet of domain standard text can produce large ontology models (figure 3a). An initial insight applying our approach to a larger text suggests that at the early stage, the size of the ontology had a relatively linear growth as text from different sections of the domain standard was analysed. As the volume of text analysed increased, a peak growth is reached when no new concepts were introduced by simply adding text from new sections of the document.

5 Conclusion and Further Work

In Requirements Management, ontologies are used to reconcile gaps in the knowledge and common understanding among stakeholders during requirement elicitation and therefore significantly improve the quality of the elicited requirements. However, a precondition of state-of-the-art ontology approaches for requirements elicitation is an existing domain ontology.

This paper identified three core properties of domain ontologies suitable for requirements elicitation. These include explicit relational expression, qualified relation identification and explicit temporal and spatial expressions. We have investigated a rule-based approach for building such domain ontologies from natural language technical documents. We first introduced bracket trailing and bridged-terms mechanism to help reduce the time and effort that is invested into pre-processing of documents so that they will be suitable for sentence-based analysis. The foundation of this approach lies in the use of NLP techniques to extract subjects and objects as concepts, while the predicate defines the relation between these extracted concepts. Association mining techniques seek to extract other types of relations that are semantically implied in the sentence, but cannot be captured by the predicate extraction process.

We evaluated the feasibility of the rule-based approach based on a real-world industrial use case by analyzing natural language text from technical standards. The study demonstrated that the rule-based approach is a viable technique for building a baseline requirements elicitation domain ontology, which can then be verified and refined by the domain expert. The study showed that requirement analysts were more prone to wrongly identifying concepts and relations to be used in domain ontologies. On the other hand, domain ontologies created by the rule-based approach and the domain expert complement each other. The evaluation provides insights into how over specification and under specification can occur for a requirements elicitation domain ontology based on analysis of natural language text.

In the short term, our further work focuses on getting more insight into the potential of using stemming/lemmatizing to reduce over specification. Future work will also focus on investigating effort reduction measures when extracting requirements elicitation domain ontologies using the rule-based approach. For instance, we seek to understand how domain experts deal with false positive alerts of ambiguous concepts during automated concept terms completion. It is also important to investigate the different modalities for baseline ontology verification and refinement. We are particularly interested in an ontology verification and refinement process that is based on understanding the risk posed by baseline ontology concepts and relations that have not been verified or refined.

References

1. Choi, F.Y.Y.: Advances in domain independent linear text segmentation. In: Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference. Morgan Kaufmann Publishers Inc., Seattle (2000)
2. Falbo, R.d.A., Guizzardi, G., Duarte, K.C.: An ontological approach to domain engineering. In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (2002)
3. Flores, J.J.G.: Semantic Filtering of Textual Requirements Descriptions. In: Natural Language Processing and Information Systems, pp. 474–483 (2004)
4. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5(2), 199–220 (1993)
5. Gruber, T.R.: Ontology. In: Liu, L., Ozsu, M.T. (eds.) Encyclopedia of Database Systems. Springer, Heidelberg (2008)
6. Ikeda, M., Seta, K., Mizoguchi, R.: Task ontology makes it easier to use authoring tools. In: Proceedings of the 15th International Joint Conference on Artifical Intelligence (1997)
7. ISO standard: Transport information and control systems -Adaptive Cruise Control Systems - Performance requirements and test procedures. 15622 (2002)
8. Kitamura, M., et al.: A Supporting Tool for Requirements Elicitation Using a Domain Ontology. In: Proceedings Software and Data Technologies (2009)
9. Kof, L.: Scenarios: Identifying Missing Objects and Actions by Means of Computational Linguistics. In: Proceedings RE 2007 (2007)
10. Kof, L.: An Application of Natural Language Processing to Domain Modelling - Two Case Studies. International Journal on Computer Systems Science Engineering 20, 37–52 (2005)
11. Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 197–211. Springer, Heidelberg (2009)
12. Kof, L.: Using Application Domain Ontology to Construct an Initial System Model. In: IASTED International Conference on Software Engineering (2004)
13. Lee, Y., Zhao, W.: An Ontology-Based Approach for Domain Requirements Elicitation and Analysis. In: Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (2006)
14. Lennard, J.: But I Digress: The Exploitation of Parentheses in English Printed Verse. Clarendon Press, Oxford (1991)
15. Liddy, E.D.: Natural Language Processing. In: Encyclopedia of Library and Information Science, 2nd edn. Marcel Decker, Inc., New York (2001)
16. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of English: the penn treebank. Comput. Linguist. 19(2), 313–330 (1993)
17. Pohl, K.: The three dimensions of requirements engineering: a framework and its applications. Inf. Syst. 19(3) (1994)
18. Shibaoka, M., Kaiya, H., Saeki, M.: GOORE: Goal-Oriented and Ontology Driven Requirements Elicitation Method. In: Hainaut, J.-L., Rundensteiner, E.A., Kirchberg, M., Bertolotto, M., Brochhausen, M., Chen, Y.-P.P., Cherfi, S.S.-S., Doerr, M., Han, H., Hartmann, S., Parsons, J., Poels, G., Rolland, C., Trujillo, J., Yu, E., Zimányie, E. (eds.) ER Workshops 2007. LNCS, vol. 4802, pp. 225–234. Springer, Heidelberg (2007)
19. Sowa, J.F.: Conceptual structures: information processing in mind and machine. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1994)

Tackling Semi-automatic Trace Recovery for Large Specifications

Jörg Leuser and Daniel Ott

Daimler AG, Group Research & Advanced Engineering,
P.O. Box 2360, 89013 Ulm, Germany
{joerg.leuser,daniel.ott}@daimler.com

Abstract. **[Context and motivation]** Traceability is not as well established in the automobile industry as it is for instance in avionics. However, new standards require specifications to contain traces. Manually creating and maintaining traceability in large specifications is cumbersome and expensive. **[Question/problem]** This work investigates whether it is possible to semi-automatically recover traceability within natural language specifications (e.g. requirement and test specifications) using information retrieval algorithms. More specifically, this work deals with large, German specifications from the automobile industry. **[Principal ideas/results]** Using optimized algorithms, we are able to retrieve most of the traces. The remaining problem is the reduction of false-positive candidate traces. **[Contribution]** We identified optimizations that improve the retrieval quality: Use of meta-data, filtering of redundant texts, use of domain language, and dynamic identification of signals.

Keywords: Traceability, Traceability Recovery, German Specifications, Large Specifications, Natural Language, Information Retrieval.

1 Introduction

Currently, there are no regulations that require the automobile industry to maintain explicit traceability through their development cycle. But this is going to change for safety critical components with the upcoming ISO 26262 [14]. Current specifications do not contain complete trace sets but will require them to be available when they are reused. Creating traceability manually is quite arduous as the specifications are large. Konrad and Gall [16] report that manually tracing 4,000 user requirements is quite a challenge. The US DoD reportedly spends four percent of life cycle costs on requirements traceability [22].

Other researchers proposed semi-automatic solutions that successfully apply information retrieval algorithms in order to create and maintain traceability after-the-fact [1,6,13,19,20,25]. However, publications of existing research focused on small, English specifications ($< 1{,}000$ elements) while the specifications in the German automobile industry are larger and often in German. The size and language pose a problem for the semi-automatic methods. We propose and describe optimizations to decrease the impact of these obstacles. The next section will describe the problem and current solutions in more detail.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 203–217, 2010.

2 The Problem

In the automotive industry, three main abstraction layers exist: Vehicle, systems, and components. For each of these layers, requirement and test specifications exist. Specifying a car easily requires around 100 system and multiple hundred component specifications. The classification of Regnell et al. [24] places many specifications into the category 'large-scale RE' (order of magnitude of 1,000 requirements). Some specifications contain up to 50,000 individual elements (approximately 2,000 pages) and therefore fall in the category 'very large-scale RE' (order of magnitude of 10,000 requirements). The sum of all specifications for a system easily reaches this category.

Two main kinds of traceability are important: Traces within or between requirement specifications and traces between requirement and test specifications. Their creation and maintenance is quite a challenge. Semi-automatic methods using information retrieval (IR) algorithms promise to reduce this effort. However, these methods are not ready for industry use as they do not reduce the manual effort enough. We outlined a number of challenges [17] for traceability in our context. Two of these directly affect the use of semi-automatic methods: The size and language of specifications. The optimizations described in this paper (Sect. 5) mainly address the size of specifications.

Research on the use of IR methods for recovery of traceability focused on three IR models: Vector Space Model (VSM) [1,6,13], Probabilistic Model [6], and Latent Semantic Indexing (LSI) [9,13,19,20,25]. The datasets used for validation are comparably small and mainly in English. Winkler's [25] dataset 'AB' is reasonably large. It contains hundreds of elements on two abstraction layers, but only an unspecified subset is used. As De Lucia et al. [9] point out and we could confirm [18], with the size of the dataset, the number of candidate traces grows fast and the ratio between actual links and false positives deteriorates rapidly. The language is another problem we face: Braschler and Ripplinger [5] showed for classic IR that the German language needs different preprocessing than for example English. We found this also to be true for the search for traceability [18]. German grammar is more complex than English grammar. This leads to a larger number of word forms that have to be dealt with. German also allows nearly unrestricted compounding of words. Compounding is widely used for technical terms. For example 'Dachbedieneinheit' (overhead control unit) is compounded of three words. The English equivalent consists of three individual words. These language properties have to be dealt with and worsen the precision of the results.

For a better grasp of the problem, we analyzed a subset of German requirements specifications of the Daimler AG. The analyzed 70 system and 106 component specifications contained a total of 62,116 different words. The system specifications have an average length of 441 requirements (15 words each), the component specifications of 848 requirements (16 words each). This illustrates that the real world specifications are larger than the previously published datasets. The statements about size are based on user generated meta-data which tags specification elements to be of a certain kind, e.g. 'requirement' or 'information' (see column 'object type' in Fig. 1). The quality of this meta-data varied.

2.1 The Datasets

The described optimizations to algorithms (Sect. 5) are validated with a number of datasets taken from the passenger car division of Daimler AG. The original specifications often contain multiple variants of the specified systems or components [4]. We extracted single-variant specifications out of the otherwise unaltered original specifications. Datasets were selected based on these criteria:

- Large specification, mainly written in German
- Different kind of traceability that is different kinds of specifications (system → system, system → component, system → test)
- Availability of a (reference) trace set (which must have been created manually).

Based on the criteria, two datasets were selected for analysis: Dataset **OLC** consists of a system specification for an **O**utside **L**ight **C**ontrol. Dataset **LSC** consists of different specifications of a **L**oading **S**pace **C**over: A system specification, a test and two component specifications providing parts of the functionality.

Where needed, we extended the available trace sets towards reference sets. This task took about 34 hours. Table 1 describes the datasets. The 'other' category contains elements that have no or a different type than 'requirement' or 'heading'. CS 1 in dataset LSC has many 'other' elements due to incomplete meta-data. The component specifications in dataset LSC belong to components that have many functions. Only some of these functions are related to the system LSC, resulting in the low number of traces. A third dataset (approx. 5,500 elements in two documents) was excluded as it was initially too large for our tool.

Table 1. Descriptive statistics of the datasets

		OLC	LSC
System specification (SysS)	requirements	2,095	109
	headings	1,166	65
	other	30	83
Component specification 1 (CS 1)	requirements	–	61
	headings	–	915
	other	–	1,181
Component specification 2 (CS 2)	requirements	–	756
	headings	–	301
	other	–	241
Test specification (TS)	test cases	–	18
	test steps	–	52
	other	–	27
Reference traceability set			
SysS ↔ SysS		1,109	67
SysS ↔ CS 1		–	14
SysS ↔ CS 2		–	6
SysS ↔ Test		–	27

3 Information Retrieval

The goal of the information retrieval (IR) algorithms is to retrieve the desired data – in our case correct traces – without retrieving unwanted data. The search for results is based on a four-step process: Preprocessing, application of algorithms, creation of a candidate trace list, and its inspection by a human analyst.

We focus on the vector space model (VSM) as this is the IR model that our research tool is based on. We chose the VSM as the probabilistic model as alternative currently does not show an overall superiority [3, p. 34]. In the preprocessing stage, the document's raw texts (e.g. requirements) are tokenized. Common further steps are removal of stop words (words not helpful for the retrieval) and stemming. Due to the large number of compounds, the decomposition of such words is beneficial for German texts [18]. The result of the preprocessing stage is a list of index terms $\{t_1, ..., t_n\}$.

In the VSM, documents (e.g. individual requirements) are interpreted as vectors. Each index term t_i represents a dimension. Each document d_j is transformed into a vector $d_j = \{freq_{t_1}, ..., freq_{t_n}\}$. $freq_{t_i}$ is the number of occurrences of index term t_i in d_j. All document vectors build the so called term-document matrix $A_{ij}(freq_{t_i,j})$. This matrix can be processed differently depending on the algorithm, resulting in matrix $B_{ij}(w_{i,j})$ with term-weights instead of frequencies.

The similarity of two documents can be calculated as the cosine between the two document vectors in B_{ij}. The assumption is that the more similar the documents (smaller angle), the more likely a trace exists. Creating the candidate trace list therefore consists of calculating the similarities and ordering the results. For practical reasons, the candidate trace list only contains document pairs that have a similarity larger than the so called cutoff point. The quality of the list can be determined using two measures: recall and precision. Recall measures how well the correct results (traces of the reference trace set) are retrieved. Precision measures the amount of correct traces in the candidate list. We use generalized forms for multiple queries like for example De Lucia et al. [7] do as well:

$$\text{Recall} = \frac{\sum_i |\text{correct traces}_i \cap \text{found traces}_i|}{\sum_i |\text{correct traces}_i|} \tag{1}$$

$$\text{Precsion} = \frac{\sum_i |\text{correct traces}_i \cap \text{found traces}_i|}{\sum_i |\text{found traces}_i|} \tag{2}$$

Hayes et al. [12] argue that recall $> 60\%$ is acceptable, $> 70\%$ good and $> 80\%$ excellent. For precision, $> 20\%$ is acceptable, $> 30\%$ good and $> 50\%$ excellent. Like De Lucia et al. [8], we share this assessement. This scale demonstrates that a complete semi-automatically retrieved trace set is currently unlikely. The already mentioned semi-automatic solutions (e.g. [8,12,25]) typically reach good to excellent recall with acceptable to good precision for small specifications.

Term Frequency/Inverse Document Frequency (tf/idf). A weighting algorithm in the VSM is the so called tf/idf algorithm [3]. It calculates the weights

$w_{i,j}$ in the term-document matrix based on two factors: the term frequency factor tf (Eq. 3) and the inverse document frequency factor idf (Eq. 4). $tf_{i,j}$ indicates how well t_i characterizes the document. idf_i represents the importance of t_i in the whole document corpus. k = index of t with maximum occurrence, N = number of documents, n_i = number of documents containing index term t_i.

$$tf_{i,j} = \frac{freq_{i,j}}{freq_{t_k,j}}, freq_{t_k,j} = max_i freq_{t_i,j} \quad (3) \qquad idf_i = \log \frac{N}{n_i} \quad (4)$$

The final weight for index term t_i in document d_j is calculated as follows:

$$w_{i,j} = tf_{i,j} \cdot idf_i \quad (5)$$

4 The TraceTool

The research tool we use is called TraceTool which is described in more detail in [17]. The TraceTool is able to access 'live' data in Doors databases. It implements the two IR algorithms tf/idf and LSI [3]. Using those algorithms, a list of candidate traces is created. In this paper, we just employ the tf/idf algorithm. For a more intuitive similarity measure for the user, the raw cosine similarities are transformed into the interval [0%,100%]. This measure is called trust level.

In order to evaluate the proposed optimizations (Sect. 5), we introduced a functionality to automatically change and measure different configurations. A so called *Measurement Run Set* defines what optimizations to activate and what parameters to set. It executes the processing and records measures like precision and recall on different cutoff-point levels. Thus, a large number of configurations can be tested automatically. This is important as (pre-)processing a configuration of the larger datasets takes between a quarter of an hour up to 6 hours, depending on algorithm and optimizations. For example evaluating the influence of the weight of one optimization (Sect. 5) in the interval [0,10] (step size 0.1) on dataset OLC (Sect. 2.1) using tf/idf with all other parameters fixed takes approximately a day on our fastest machine. Only the thesaurus which changes the similarity calculation (Sect. 5.2) extends the processing time significantly.

5 Investigated Optimizations

We focused on the tf/idf algorithm with our optimizations. The main reason is that the correlation of changes to the algorithm (e.g. $w_{i,j}$) and changes in the results (the resulting similarities) are more comprehensible than with LSI which processes B_{ij} heavily. Furthermore, LSI could make use of the extended tf/idf by using the differently weighted term-document matrix. We investigated different kinds of optimizations: First, filters to discard candidate traces based on document text or meta-data. Secondly, the weighting of the tf/idf algorithm (Eq. 5) was modified by exploiting knowledge about individual index terms.

Some of the optimizations can be applied at different processing stages like preprocessing (earliest) or candidate trace list creation (latest). The filter depending on meta-data (Sect. 5.4) can either be applied before preprocessing (filtered documents are not included in A_{ij}) or when the candidate list is created. Results showed that the use of the optimizations depend on the dataset so we opted for the latest possibility when there was a choice. Due to this decision, the user is able to change the filters' preferences quickly. After preference changes, only the creation of the list of candidate traces has to be repeated.

5.1 Dynamic Signal Weighting (DSW)

Systems in the automotive context are distributed. Therefore, the different components have to communicate using communication buses (e.g. CAN, FlexRay). The communication is specified using (logical) signals, i.e. named information containers, which can be transferred via bus systems. Analyzing reference traces, we found signals to be an important criterion in the existence of traces. Therefore, we reasoned that supplying this knowledge could improve the retrieval results. Although the exact signal names are not known beforehand, we are able to dynamically identify them (via a regular expression) as they follow a known naming scheme. Knowing the signals, we are able to attach an additional weight x_i to such signals; sw: parameter to be set; anticipated optimum value at $sw > 1$:

$$w_{i,j} = tf_{i,j} \cdot idf_i \cdot x_i \quad (6) \qquad x_i = \begin{cases} sw_i, \text{if } t_i \in \text{signals} \\ 1, else \end{cases}$$

Signals with Value Assignment (VA). Hayes et al. [11] found that incorporating phrases, that is index terms with more than one word, can improve retrieval results. We found signals with value assignment that seem to be similar to phrases: For example, requirements might contain '$LoBM\_FLT = 0$' which differs from '$LoBM\_FLT = 1$' although the signal name is the same. The identification of such index terms is done during preprocessing. From there on, the index terms are treated like 'normal' index terms.

5.2 Domain Language (D)

Although the idea behind tf/idf is that more important index terms get a higher term-weight, we thought that having exact knowledge about individual terms could have advantages. In the following sections, we will describe different approaches to handle domain language.

Word Classes. It is evident that not all words are created equally: Some have more semantic than others. Stopwords, for example, are not seen as helpful for the retrieval task and are therefore discarded. We propose two additional word classes besides stopwords and 'normal' words: weak words (mostly as commonly used in requirements engineering, e.g. [10]: 'oft' (often), 'gering' (small)) and domain words (e.g. 'Temperaturregler' (thermostat)). The signals (Sect. 5.1)

could be seen as a special form of domain words. We assume that weak words contain less meaning than 'normal' words ($ww < 1$) and domain words contain more ($dw > 1$). Therefore, we created a list of domain words (D(dw), 9,648 words) and a list of weak words (D(ww), 1,039 words) out of our extensive word list we extracted during our initial analysis. We change the term-weights $w_{i,j}$ as seen in Eq. 6 with the following weight x_i; ww, dw: parameters to be set:

$$x_i = \begin{cases} ww, \text{if } t_i \in \text{weak words} \\ dw, \text{if } t_i \in \text{domain words} \\ 1, \text{else} \end{cases}$$

Thesaurus (Th). Hayes et al. [13] report improved retrieval results when a thesaurus is included into the similarity calculation of the IR algorithms. Encouraged by these results, we wanted to test the inclusion of a thesaurus. As in their solution, we built a thesaurus T with entries of the form $\langle t_k, t_l, \alpha_{kl} \rangle$ using our initial word list as basis. α_{kl} is the similarity coefficient for the two terms t_k and t_l. The created thesaurus contains 286 entries. It is applied as an extension to the basic cosine similarity in the similarity calculation as in Eq. 7:

$$sim(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \cdot w_{i,q} + \sum_{\langle t_k, t_l, \alpha_{kl} \rangle \in T} \alpha_{kl} \cdot (w_{k,j} \cdot w_{l,q} + w_{l,j} \cdot w_{k,q})}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \cdot \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \quad (7)$$

Synonym Normalization (SN). As the similarity calculation already is responsible for a large part of the processing time, we thought about a faster way to make use of a thesaurus. We found that our thesaurus contains mainly synonyms. For example different names for car components. A component in the rear of the car called SAM is sometimes referred to as 'SAM-R', 'SAM_R', and sometimes as 'SAM_REAR'. We propose to normalize such synonyms in the preprocessing stage. This means, whenever one of the synonyms is found, it is always replaced by the same synonym (e.g. always SAM-R). This replacement is faster than the solution by Hayes et al. [13]. Of course, this solution only works for synonyms.

5.3 Filtering Redundant Texts (frt)

The specifications created by Daimler are based on different templates. These templates provide structure as well as content. Not all specifications require the whole template. However, the rules require some structural elements (that is headings) to be present even when they contain no content. These headings are marked irrelevant by placing predefined, redundant texts below them. Fig. 1 illustrates such a redundant text. Traces between such texts are found with a high confidence as the texts are identical. Such traces are obviously unwanted. Therefore, we allow filtering candidate traces when at least one end contains such a text. Preliminary results showed other filterable texts: Introductory texts to listings like 'The following error messages have to be processed:'.

Fig. 1. Illustration of a heading followed by a redundant placeholder text. The texts were translated for illustration purposes.

Filtering Empty Headings (feh) The filtering of empty headings is an extension of the previous filter. While specifications may contain empty headings, that is headings that do not contain any child elements, the main case of this filter is another one: When redundant texts in a section mark it as not relevant, the heading of this section also becomes unimportant. It may not be 'physically' empty but semantically. An example is the heading in Fig. 1. The proposed filter will remove traces to such headings from the candidate link list.

5.4 Filtering According to Meta-data (fmd)

The elements in specifications do not only contain texts but also a number of meta-data. Such meta-data is for example a classification of the different elements (see column 'object type' in Fig. 1). Requirement specifications may group its content into requirements, headings, and explanations. This proposed filter removes candidate traces to defined types of elements as for example only traces to headings and requirements are wanted.

6 Results

Our prior work [18] indicates better recall when stemming and decomposition is activated for German specifications. Therefore, all measurements were done with those two steps activated. Additionally, two filters were activated: The first filter removes candidate traces between sibling elements, i.e. elements that share the same parent (e.g. a heading). The second filter removes candidate traces where one end of the trace is a direct parent of the other end, for example a trace between the heading and the requirement in Fig. 1 would be removed. The two filters are based on the knowledge that these relationships are already documented by the document structure. Both filters have shown an increase in precision with a minimal loss of recall [18]. These filters are only effective when searching within one specification. When a dataset contained more than one specification we created pairwise subsets.

To find the best possible results for each optimization we used our 'Measurement Run Sets' (Sect. 4) to iterate through many possible parameter values (where available). We opted for 0.1 as step size for the parameters. The search space was [0,10]. It was extended to 20 when the results indicated, that higher

parameter values would produce better results. Our search space produced a large number of results. The best result was picked by the following schema:

- Take the results with the 15% best recall values in low-confidence traces (5-25% cutoff point; < 5% was not measured)
- Rank the selected results according to the slope of the regression line and take the result with the slowest decline
- When the last selection step resulted in more than one possible result, we looked at the precision in the same way

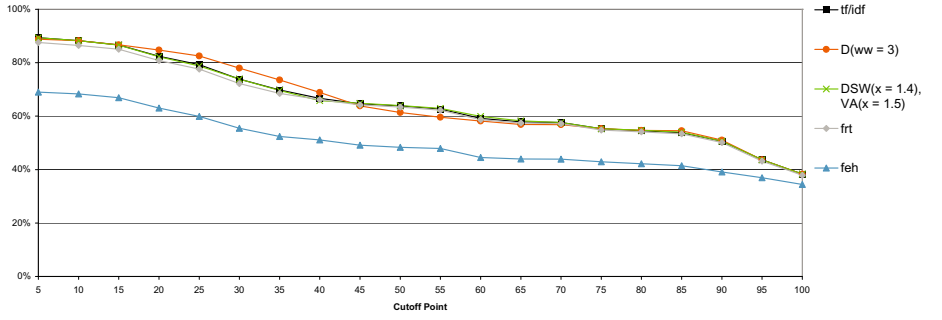
This selection scheme is based on a couple of assumptions: First, the most important task of the TraceTool is to recover traces, hence the focus on recall. Secondly, the curve with the best maximum recall might deteriorate worse than curves with slightly less maximum recall. We believe that a good recall over all cutoff points (slow deterioration) is preferable over partially good results (e.g. only good at the cutoff point that yields the maximum recall). This way, an analyst can expect similar recall at all cutoff points. As we experienced unacceptable precision values in our datasets before, we opted for precision as the last factor only. When applying our selection schema, we checked, if promising results were discarded by the first selection step. When such a result existed it was manually added. This was the case for the weak word optimization in dataset LSC in its subset Sys → CS 1.

For the filtering approaches (Sect. 5.3, 5.4), we adapted the filters to the datasets. Additional dataset specific values were included. The filter for redundant texts originally contained 10 predefined texts but was extended to up to 35 entries.

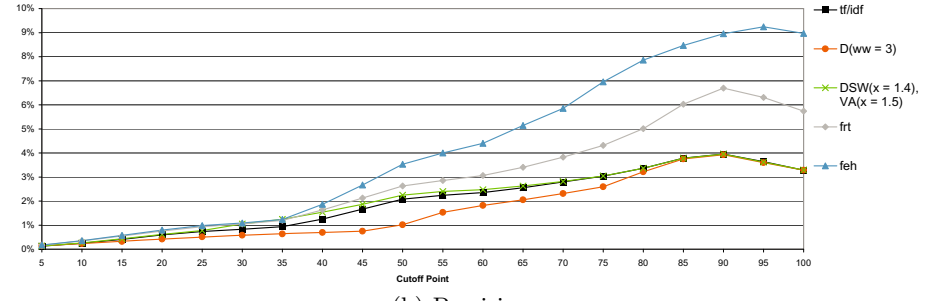
Table 2. Results for dataset OLC. Traces within the system specification.

Optimization	recall		precision		candidate traces	
	max	Ø	max	Ø	max	Ø
tf/idf	89.36%	64.91%	3.95%	1.99%	800,780	111,105
+DSW(x=1.4)	89.36%	64.97%	3.95%	2.04%	778,374	107,272
+VA(x=1.5)	89.36%	64.90%	3.95%	2.06%	762,395	105,023
+DSW(x=1.4)+VA(x=1.5)	89.36%	64.93%	3.95%	2.08%	740,177	102,394
+D(ww=3.0)	88.82%	65.29%	3.93%	1.67%	704,469	130,445
+D(dw=1.1)	89.36%	64.91%	3.96%	1.98%	798,869	112,139
+D(ww=3.0, dw=1.1)	88.91%	65.17%	3.95%	1.68%	731,132	130,871
+Th	89.45%	65.35%	3.88%	1.95%	822,606	115,437
+SN	89.45%	65.01%	3.95%	1.98%	813,389	112,509
+frt	87.56%	64.05%	6.70%	2.93%	572,109	80,656
+feh	68.98%	50.05%	9.24%	4.16%	442,354	59,227
+fmd	89.09%	64.86%	3.99%	2.01%	793,642	110,119

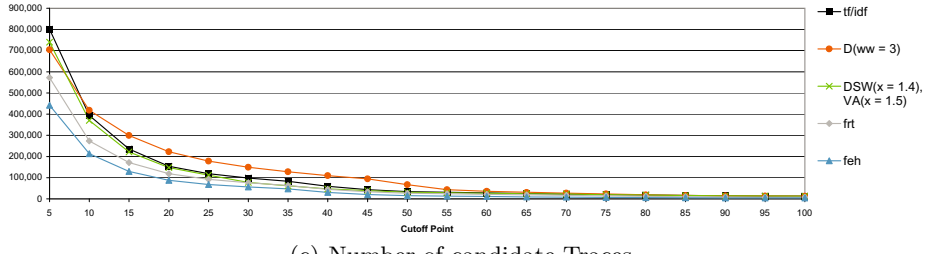
DSW: Dynamic signal weighting, VA: value assignment (Sect. 5.1); D: Domain language, ww: weak words, dw: domain words, Th: Thesaurus, SN: synonym normalization (Sect. 5.2); frt: filter redundant texts, feh: filter empty headings (Sect. 5.3); fmd: filter according to meta-data (Sect. 5.4).



(a) Recall



(b) Precision



(c) Number of candidate Traces

Fig. 2. Results for dataset OLC. Traces within the system specification.

Due to the limited space, the results of dataset OLC only are presented as graphs. Table 2 displays all results. Fig. 2(a) shows the recall over the cutoff point range 5% - 100% (low - high confidence). Fig. 2(b) shows the accompanying precision curves. Fig. 2(c) depicts the number of candidate traces. The results are reported with the best parameters selected according to our schema. For better readability, only results are included that differ from the original tf/idf or from one of the other curves visibly. Table 3 displays the results of dataset LSC with its subsets. As its system specification contains no signals with value assignment, the appropriate optimization is not applicable.

Table 3. Results for dataset LSC

Optimization	recall		precision		candidate traces	
	max	Ø	max	Ø	max	Ø
SysS → SysS						
tf/idf	80.60%	24.85%	33.33%	11.32%	4,832	570
+DSW(x=2.7)	80.60%	24.93%	33.33%	11.17%	4,578	538
+D(ww=0.1)	80.60%	26.19%	33.33%	11.94%	4,541	558
+D(dw=0.8)	80.60%	25.07%	33.33%	11.51%	4,910	573
+D(ww=0.1, dw=0.8)	80.60%	26.64%	33.33%	11.29%	4,651	558
+Th	80.60%	25.30%	25.00%	9.45%	5,185	616
+SN	80.60%	24.03%	33.33%	11.22%	4,854	572
+frt	80.60%	24.85%	33.33%	11.58%	4,685	550
+feh	80.60%	24.85%	33.33%	11.58%	4,685	550
+fmd	37.31%	13.96%	33.33%	11.75%	2,139	281
SysS → CS 1						
tf/idf	57.14%	25.36%	1.03%	0.48%	45,561	4,377
+DSW(x=14.5)	57.14%	25.36%	1.04%	0.48%	40,336	3,991
+D(ww=3.5)	57.14%	27.86%	0.87%	0.28%	44,646	7,420
+D(dw=0)	64.29%	27.14%	0.47%	0.24%	56,606	7,184
+D(ww=3.5, dw=0)	57.14%	30.36%	0.47%	0.14%	47,610	9,845
+Th	57.14%	25.36%	0.98%	0.45%	46,224	4,555
+SN	57.14%	25.00%	1.03%	0.47%	46,156	4,440
+frt	57.14%	25.36%	1.11%	0.57%	40,814	3,816
+feh	57.14%	25.36%	1.36%	0.68%	40,267	3,709
+fmd	28.57%	9.64%	0.37%	0.21%	29,579	2,904
SysS → CS 2						
tf/idf	83.33%	19.17%	0.87%	0.19%	16,908	1,524
+DSW(x=15)	83.33%	19.17%	0.88%	0.20%	15,260	1,405
+D(ww=0.8)	83.33%	19.17%	0.85%	0.19%	16,396	1,483
+D(dw=0.7)	83.33%	19.17%	0.83%	0.14%	17,452	1,592
+D(ww=0.8, dw=0.7)	83.33%	19.17%	0.85%	0.15%	17,000	1,547
+Th	83.33%	19.17%	0.69%	0.16%	17,055	1,583
+SN	83.33%	19.17%	0.85%	0.19%	17,056	1,535
+frt	83.33%	19.17%	1.19%	0.26%	15,500	1,358
+feh	83.33%	19.17%	1.30%	0.28%	15,270	1,333
+fmd	16.67%	1.67%	0.03%	0.00%	9,150	839
SysS → TS						
tf/idf	74.07%	23.52%	100.00%	39.64%	1,853	209
+DSW(x=5.2)	74.07%	24.63%	100.00%	33.01%	1,664	185
+D(ww=0.8)	74.07%	23.15%	100.00%	39.39%	1,763	200
+D(dw=1.6)	74.07%	23.33%	100.00%	42.68%	1,600	212
+D(ww=0.8, dw=1.6)	74.07%	23.52%	100.00%	42.83%	1,542	208
+Th	74.07%	23.52%	100.00%	39.65%	1,870	210
+SN	74.07%	23.52%	100.00%	39.58%	1,857	210
+frt	74.07%	23.52%	100.00%	39.63%	1,792	205
+feh	74.07%	23.52%	100.00%	39.73%	1,784	203
+fmd	62.96%	21.85%	100.00%	39.67%	1,334	167

7 Discussion

There is only one interpretation: The results are not adequate for use in industry. However, there is light: With most of our datasets, we were able to retrieve most of the reference traces. The remaining problem is the precision. Although we reach 10% precision in some sets, this is not enough when compared with the huge amount of candidate traces that have to be analyzed. These results are due to the large size of the datasets in terms of semi-automatic trace retrieval. The precision in the smaller subsets (see e.g. SysS → TS in Tab. 3) is considerably better. For even larger datasets not uncommon in industry, even worse precision results are expected. Although the results show mostly little impact on precision and recall in absolute terms, these changes become important through the size of our datasets: The number of candidate traces needing inspection show large differences. If we take dataset LSC with SysS → CS 1, we see an increase of the average precision of 0.09 only between tf/idf and the filtering of redundant texts (frt). However, in absolute terms, the difference is more obvious: 561 traces less (−12.8%) need inpection per cutoff point – without loss of recall.

The different optimizations performed unequally. Dynamic signal weighting (DSW) either affected the recall mildly positively or not at all. It always lead to fewer candidate traces. This might be due to the fact that signals are an important factor for the existence of traces. The additional consideration of signals with value assignment (DSW+VA) did not increase the recall but improved the precision further. However, the optimal weight parameter varies quite a bit.

The handling of domain vocabulary in form of weak and domain words was not very successful. Although weak words (D(ww)) helped to increase the recall in most datasets, the precision worsened. In case SysS → SysS of dataset LSC only, the precision was improved. We expected the parameter for weak words to be < 1 as weak words should be semantically less important than other words. However, this was not always the case. Similar unexpected 'behavior' of weak words was observed when the quality of automotive specifications was rated by requirements engineers: The higher the amount of weak words, the better the perceived quality [26]. Domain words (D(dw)) improved the recall most of the time while always increasing the number of candidate traces. The combination of both word classes did not perform well as especially the precision suffers.

The use of a thesaurus improved or kept the recall. The precision was reduced in all but one set. The synonym normalization (SN) – a simpler form of the thesaurus (Th) – performed slightly worse on the recall side. However, the precision was not as badly influenced as with the thesaurus. For large datasets, SN seems to be the better choice as it is faster.

The filtering of redundant texts (frt) is especially helpful for the search of traces within one document, or more specifically within one document template. This might be due to the fact that the different templates use different redundant texts. It should be noted that beside in dataset OLC, the removal of redundant texts did not diminish the recall but always improved the precision. The extension that removes (semantically) empty heading (feh) can improve the precision further. As with frt, the recall is reduced in dataset OLC only, while the precision is doubled.

The filtering according to meta-data (fmd) resulted in mixed results. They depend heavily on the quality of available meta-data. Precision was improved in about half our measurements. Recall was always reduced what indicates weaknesses in the available meta-data. Notable is the result in dataset LSC between the SysS and CS 2: nearly all reference traces were filtered out, rendering this filter useless in this set.

8 Related Work

Research into optimizations for semi-automatic methods recovering traces lead to different approaches and results.

Multiple researchers studied finding traces between artifacts without common language. McMillan et al. [21] propose identifying traces by taking an indirection: When two documents are related to the same part of code, it is assumed that the documents also are related to each other. Asuncion and Taylor [2] propose deriving candidate traces by monitoring the way the user interacts with artifacts. They reason that when a user concurrently or sequentially modifies artifacts, they might be related. Their approach is based in the e-Science domain but should be transferable to model based specifications.

Contrary to model based development with code generation, model based specifications are not very common in the automobile domain. Should this change, the proposals of De Lucia et al. [9] and Cleland-Huang et al. [6] promise to retrieve traces between requirements and models. Kof [15] uses natural language processing to build message sequence charts and automata from textual scenarios. He therefore provides means to automatically formalize parts of natural language specifications. Such transformations allow tracing the different representations.

For later phases of development, Marcus and Maletic [20] show that traces into code can be retrieved. Ratanotayanon et al. [23] support tracing between feature-descriptions and code and are able to automatically update traces on code changes. Winkler [25] proposes to extend the IR approaches in order to reuse the analyst's decision about candidate traces even when an artifact changes. Although his approaches are beneficial when 'good' decisions were made, they also preserve 'bad' decisions.

9 Conclusions

Finding traces in some sets is like looking for a needle in a haystack. For example, finding 6 traces between the LSC system specification and CS 2 in about 1,500 elements is not that easy. Unsurprisingly, sets with traces into component specifications have bad precision results. One of the reasons might be that just a very small part of the component specifications is relevant for the system. Obviously, only traces to the relevant parts should be retrieved. Therefore, we tested removing the irrelevant parts. The results improved – as expected – enormously: For example for SysS → CS 1, the recall went up to 28.57% and precision to 4.27%. The precision of the new results of the original tf/idf is nearly 10-times as good

as the previously achieved 0.48%. As our goal is to work with unaltered specifications, we propose extending the filtering mechanisms or the user interaction in a way to facilitate finding traces between subparts of specifications.

The use of dynamic signal weighting always reduced the number of candidate traces without loss or even with gain in recall. If applicable, the consideration of signals with value assignment also has positive effects. The introduction of two word classes 'weak words' and 'domain words' with additional weights generally improved the recall but often at the cost of reduced precision. The same is true for the use of a thesaurus or the faster form of synonym normalization.

It is evident to us that the current results are not good enough for an industrial application. However, we also see that we were able to improve the number of candidate traces needing inspection. The filtering approaches based on redundant texts, empty headings, and meta-data generally improved the precision. Often, it is possible to remove false positive candidate traces without any loss of recall. As this depends on the dataset, our decision to apply the filters as late as possible in the processing chain seems to be correct. Hence the analyst can decide at analysis time what filters to activate.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A.: Identifying the Starting Impact Set of a Maintenance Request: a Case Study. In: Proceedings of the Fourth European Software Maintenance and Reengineering, pp. 227–230 (2000)
2. Asuncion, H.U., Taylor, R.N.: Capturing Custom Link Semantics among Heterogeneous Artifacts and Tools. In: ICSE Workshop on Traceability in Emerging Forms of Software Engineering (2009)
3. Baeza-Yates, R., Ribeiro-Neto, B.A.: Modern Information Retrieval, reprint edn. Pearson Addison-Wesley (2006)
4. Boutkova, E.: Variantendokumentation in Lastenheften: State-of-the-Practice (Variant Documentation in Requirement Specifications). In: Systems Engineering Infrastructure Conference (November 2009)
5. Braschler, M., Ripplinger, B.: How Effective is Stemming and Decompounding for German Text Retrieval? Information Retrieval 7(3-4), 291–316 (2004)
6. Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability. In: 13th IEEE International Conference on Requirements Engineering, pp. 135–144. IEEE CS, Los Alamitos (2005)
7. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an Artefact Management System with Traceability Recovery Features. In: 20th IEEE International Conference on Software Maintenance, pp. 306–315. IEEE CS, Los Alamitos (2004)
8. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Can Information Retrieval Techniques effectively Support Traceability Link Recovery? In: 14th IEEE International Conference on Program Comprehension, pp. 307–316 (2006)
9. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods. ACM Transactions on Software Engineering and Methodology 16(4), 13 (2007)
10. Dreher, M.: Konstruktive und analytische Methoden zur Qualitätssicherung von Anforderungen in der Softwareentwicklung (Constructive and Analytical Methods for Quality Assurance of Requirements in SW Development). Stuttgart Media University, Diplomarbeit (2004)

11. Hayes, J.H., Dekhtyar, A., Osborne, J.: Improving Requirements Tracing via Information Retrieval. In: 11th IEEE International Requirements Engineering Conference, pp. 138–147 (2003)
12. Hayes, J.H., Dekhtyar, A.: Humans in the Traceability Loop: Can't Live with 'em, Can't Live without 'em. In: Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 20–23. ACM, New York (2005)
13. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. IEEE Transactions on Software Engineering 32(1), 4–19 (2006)
14. ISO/DIS 26262: Road Vehicles – Functional Safety. ISO (2009)
15. Kof, L.: Translation of Textual Specifications to Automata by Means of Discourse Context Modeling. In: Glinz, M., Heymans, P. (eds.) REFSQ 2009. LNCS, vol. 5512, pp. 197–211. Springer, Heidelberg (2009)
16. Konrad, S., Gall, M.: Requirements Engineering in the Development of Large-Scale Systems. In: 16th IEEE International Conference on Requirements Engineering, pp. 217–222 (2008)
17. Leuser, J.: Challenges for Semi-automatic Trace Recovery in the Automotive Domain. In: ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 31–35 (May 2009)
18. Leuser, J.: Herausforderungen für halbautomatische Traceability-Erkennung (Challenges for Semi-automatic Trace Recovery). In: Systems Engineering Infrastructure Conference (November 2009)
19. Lormans, M., van Deursen, A.: Can LSI Help Reconstructing Requirements Traceability in Design and Test? In: Proceedings of the Conference on Software Maintenance and Reengineering, pp. 47–56. IEEE CS, Los Alamitos (2006)
20. Marcus, A., Maletic, J.I.: Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing. In: 25th International Conference on Software Engineering, 3rd edn., pp. 3–10 (2003)
21. McMillan, C., Poshyvanyk, D., Revelle, M.: Combining Textual and Structural Analysis of Software Artifacts for Traceability Link Recovery. In: ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 41–48 (May 2009)
22. Powers, T., Stubbs, C.: A Study on Current Practices of Requirements Traceability in Systems Development. Masterthesis, Naval Postgrad. School Monterey CA (1993)
23. Ratanotayanon, S., Sim, S.E., Raycraft, D.J.: Cross-Artifact Traceability Using Lightweight Links. In: ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 57–64 (May 2009)
24. Regnell, B., Svensson, R.B., Wnuk, K.: Can we Beat the Complexity of very Large-Scale Requirements Engineering? In: Paech, B., Rolland, C. (eds.) REFSQ 2008. LNCS, vol. 5025, pp. 123–128. Springer, Heidelberg (2008)
25. Winkler, S.: Trace Retrieval for Evolving Artifacts. In: ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp. 49–56 (May 2009)
26. Yakoubi, R.: Empirische Bewertung von Qualitätsindikatoren für Anforderungsdokumente (Empirical Assessment of Quality Indicators for Requirement Specifications). Ulm University, Diplomarbeit (2009)

Ambiguity Detection: Towards a Tool Explaining Ambiguity Sources

Benedikt Gleich[1], Oliver Creighton[2], and Leonid Kof[3]

[1] Fakultät für Angewandte Informatik, Universität Augsburg,
Universitätsstr. 6a, D-86159 Augsburg, Germany
benedikt.nikolaus.gleich@student.uni-augsburg.de
[2] Siemens AG, Corporate Technology, Otto-Hahn-Ring 6, D-81730 München
oliver.creighton@siemens.com
[3] Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748, Garching bei München, Germany
kof@informatik.tu-muenchen.de

Abstract. [**Context and motivation**] Natural language is the main representation means of industrial requirements documents, which implies that requirements documents are inherently ambiguous. There exist guidelines for ambiguity detection, such as the Ambiguity Handbook [1]. In order to detect ambiguities according to the existing guidelines, it is necessary to train analysts.

[**Question/problem**] Although ambiguity detection guidelines were extensively discussed in literature, ambiguity detection has not been automated yet. Automation of ambiguity detection is one of the goals of the presented paper. More precisely, the approach and tool presented in this paper have three goals: (1) to automate ambiguity detection, (2) to make plausible for the analyst that ambiguities detected by the tool represent genuine problems of the analyzed document, and (3) to educate the analyst by explaining the sources of the detected ambiguities.

[**Principal ideas/results**] The presented tool provides reliable ambiguity detection, in the sense that it detects four times as many genuine ambiguities as than an average human analyst. Furthermore, the tool offers high precision ambiguity detection and does not present too many false positives to the human analyst.

[**Contribution**] The presented tool is able both to detect the ambiguities and to explain ambiguity sources. Thus, besides pure ambiguity detection, it can be used to educate analysts, too. Furthermore, it provides a significant potential for considerable time and cost savings and at the same time quality improvements in the industrial requirements engineering.

Keywords: requirements analysis, ambiguity detection, natural language processing.

1 Requirements Documents Are Ambiguous

The overwhelming majority of requirements documents are written in natural language, as the survey by Mich et al. shows [2]. This implies that requirements are imprecise, as precision is difficult to achieve using mere natural language as the main presentation means. In software development, the later an error is found, the more expensive its

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 218–232, 2010.

correction is. Thus, the imprecision of requirements should be detected early in the development process.

Ambiguity (i.e., the possibility to interpret one phrase in several ways) is one of the problems occurring in natural language texts. An empirical study by Kamsties et al. [3] has shown that "... ambiguities are misinterpreted more often than other types of defects; ambiguities, if noticed, require immediate clarification". In order to systematize typical ambiguous phrases, Berry et al. introduced the Ambiguity Handbook [1]. A tool that detects ambiguities listed in the handbook surely contributes to early detection of problematic passages in requirements documents. According to Kiyavitskaya et al. [4], a tool for ambiguity detection should not only detect ambiguous sentences, but also explain, for every detected sentence, what is potentially ambiguous in it. Such a tool is presented in this paper.

Contribution: The tool described in the presented paper is a big step towards the ambiguity detection tool satisfying the requirements by Kiyavitskaya et al: this tool is able not only to detect ambiguities but also to explain ambiguity sources. When detecting ambiguities, it basically relies on a `grep`-like technique, which makes it highly reliable, applicable to different languages, and independent from error-prone natural language parsing. For every detected ambiguity the tool provides an explanation why the detection result represents a potential problem. Furthermore, due to web-based presentation and a lightweight linguistic engine on the server side, the tool is fast and highly portable, which makes it applicable for real projects. Therefore, it can cause considerable time and cost savings while at the same time enabling higher quality, as it simplifies early detection of potentially critical errors.

Outline: The remainder of the paper is organized as follows: First, Section 2 sketches part-of-speech tagging, the computational linguistics technique used in our tool. Then, Section 3 introduces the types of ambiguities that can be detected by our tool. These types include ambiguity classes introduced in the Ambiguity Handbook and ambiguity classes derived from writing rules used internally at Siemens. Section 4 presents the tool itself, especially the technique used to detect ambiguities and the presentation of the detected ambiguities to the tool user. Section 5 provides the results of the tool evaluation. Finally, Sections 6 and 7 present an overview of related work and the summary of the paper, respectively.

2 Computational Linguistics Technique: Part-of-Speech Tagging

Part-of-speech (POS) tagging marks every word of a given sentence with one of the predefined parts-of-speech (substantive, adjective, ...) by assigning a POS tag. For example, the words of the sentence "Failure of any other physical unit puts the program into degraded mode" are marked in the following way: `Failure°NN°failure of°IN°of any°DT°any other°JJ°other physical°JJ°physical unit°NN°unit puts°VBZ°put the°DT°the program°NN°program into°IN°into degraded°VBN°degrade mode°NN°mode`. Here, `NN` means a noun, `DT` a determiner, `JJ` an adjective, `VBZ` a verb, and `IN` a preposition. Additionally to part-of-speech tags, the tagger can provide the basic form for every word, as in the example presented above. Basic forms will be important for us to detect passive voice,

where we will look for the verb "be". Following tags are the most important ones in the context of the presented work: (1) any tag starting with "VB", identifying different verb forms, (2) tag "VBN", identifying verbs in the past participle form ("been", "done"), (3) any tag starting with "NN", identifying different noun forms, (4) tag JJ, identifying adjectives, and (5) tag RB, identifying adverbs. Complete information on tag meanings can be found in the official specifications of tagsets [5,6]. Tagging technology is rather mature: there are taggers available with a precision of about 96%, such as the TreeTagger [7,8], used in the presented work. The applied tagger (TreeTagger) provides support for English, German, and further languages, and thus allows to extend the presented work to really multilingual ambiguity detection.

3 Types of Ambiguities Detected by the Tool

The Ambiguity Handbook [1] lists several types of ambiguities, namely lexical, syntactic, semantic, and pragmatic ambiguities. Additionally, the Ambiguity Handbook states vagueness and language errors as further sources of problems. All these sources of problems are briefly introduced below. The citations in the below list are taken from the Ambiguity Handbook.

Lexical ambiguity: "Lexical ambiguity occurs when a word has several meanings." This can be the case, for example, when one word has several meanings (like "green" meaning "of color green" or "immature"), or two words of different origin come to the same spelling and pronunciation (like "bank" meaning "river bank" and "bench").

Syntactic ambiguity: "Syntactic ambiguity, also called structural ambiguity, occurs when a given sequence of words can be given more than one grammatical structure, and each has a different meaning." This can be the case, for example, when the sentence allows different parse trees, like "small car factory" that can mean both "(small car) factory" and "small (car factory)".

Semantic ambiguity: "Semantic ambiguity occurs when a sentence has more than one way of reading it within its context although it contains no lexical or structural ambiguity." This can be the case, for example, when several quantifiers occur in the same sentence, like in "all citizens have a social security number" that can be interpreted in two ways:

- every citizen has an individual social security number:
  $\forall x.citizen(x) \Rightarrow \exists y.social\_security\_number(y) \wedge has(x, y)$
- all citizens have the same social security number:
  $\exists y.(social\_security\_number(y) \wedge \forall x.(citizen(x) \Rightarrow has(x, y)))$

Pragmatic ambiguity: "Pragmatic ambiguity occurs when a sentence has several meanings in the context in which it is uttered." This can be the case when references occurring in the text can be resolved in several ways. For example, "they" in "The trucks shall treat the roads before they freeze" can refer both to the roads and to the trucks.

Vagueness: Vagueness occurs when a phrase has a single meaning from grammatical point of view, but still leaves room for interpretation, when considered as a requirement. "The system should react *as fast as possible*" provides an example of such a vague phrase.

Language error: Language errors represent grammatically wrong constructions, like "Every light has their switch." The above construction can be interpreted in different ways, both as "Every light has its own switch" and as "All lights have their common switch" and cause problems later.

Independently from the ambiguity type, we can apply ambiguity detection on the same four levels, namely lexical, syntactic, semantic, and pragmatic. These are the levels traditionally used in natural language processing, cf. [9]. Analysis tasks and result types for every kind of analysis are sketched in Table 1. A survey performed in our previous work [10] shows that solely lexical and syntactic analyses are possible at the moment for fully-fledged English. If the grammar used in the text can be restricted to a certain subset of English, semantical analysis becomes possible, too. Attempto Controlled English [11] gives an example of such a restricted language and a processing tool for this language. Pragmatic analysis is not possible yet. In order to keep our tool applicable to different documents, written without any grammatical restrictions, as well as to make the tool efficient, we focus on lexical and lightweight syntactical analysis, based on part-of-speech tagging. The applied analysis rules are presented below in Section 4.

Table 1. Classification of text analysis techniques

Analysis type	**Analysis tasks**	**Analysis results**
lexical	identify and validate the terms	set of terms used in the text
syntactic	identify and classify terms, build and validate a domain model	set of terms used in the text and a model of the system described in the text
semantic	build a semantic representation of every sentence	logical representation of every sentence, formulae
pragmatic	build a representation of the text, including links between sentences	logical representation of the whole text, formulae

The patterns for ambiguity detection have been extracted from the Ambiguity Handbook and an Siemens-internal guidelines for requirements writing. The Ambiguity Handbook lists a total of 39 types of ambiguity, vagueness or generality. Some of these patterns were not integrated into our tool: 4 out of 39 types were isolated cases, i.e., examples of ambiguous expressions without explicit statements, which linguistic patterns can be used to identify the ambiguity. "Mean water level" is an example of such an expression: to make it unambiguous, it is necessary to define precisely, how "mean" is determined, but we cannot generalize this expression to an ambiguity detection pattern. 7 further ambiguities are on a semantic or pragmatic level and are not amenable to state-of-the-art computational linguistics. Elliptic ambiguities like "Perot knows a richer man than Trump" provide an example of such a high level ambiguity. Lastly, ambiguities in formalisms (counted as one of 39 ambiguity types too) were not included in our tool, as we aim at the analysis of requirements written in natural language.

The remaining 27 patterns were integrated into our tool. In addition, all 20 patterns from the Siemens guidelines could be integrated, as they all can be easily detected on lexical or syntactic level. 9 out of 20 Siemens patterns are already covered by 8 patterns

Table 2. Ambiguity patterns with source and level of detection. Sources: AH=Ambiguity Handbook, S=Siemens

Ambiguity	Source	Ambiguity level	Level of detection
"up to", without explicit "including/excluding"	AH	semantic	syntactic
they	AH	pragmatic	lexical
everybody followed by their/its	AH, S	semantic	syntactic
Ambiguous words like include, minimum, or, ...	AH	semantic, pragmatic	lexical
Vague words like acceptable, easy, efficient...	AH	semantic, pragmatic	lexical
Dangerous plural: all, each, every...	AH, S	semantic, pragmatic	syntactic
Dangerous plural with ambiguous reference (e.g. every ... a)	AH, S	semantic, pragmatic	syntactic
Both at the beginning of a sentence	AH	semantic, pragmatic	syntactic
many, few	AH	semantic, pragmatic	lexical
only, also, even	AH, S	syntactic	lexical
otherwise, else, if not	AH	semantic, pragmatic	lexical
not	AH, S	semantic, pragmatic	lexical
not because	AH	semantic, pragmatic	lexical
"and" and "or" in the same sentence	AH, S	syntactic, semantic	syntactic
until, during, through, after, at	AH	semantic, pragmatic	lexical
could, should, might	S	semantic, pragmatic	lexical
usually, normally	S	semantic, pragmatic	lexical
actually	S	semantic, pragmatic	lexical
100%, all errors	S	pragmatic	lexical
he, she, it	AH, S	pragmatic	lexical
brackets	S	syntactic, semantic, pragmatic	lexical
Slashes	S	syntactic, semantic, pragmatic	lexical
tbd, etc	S	semantic, pragmatic	lexical
fast	S	semantic, pragmatic	lexical
passive	S	semantic, pragmatic	syntactic, with part-of-speech tags
"this" (ambiguous reference)	AH, S	semantic, pragmatic	lexical
vague or ambiguous adjectives	AH	lexical, semantic and pragmatic	syntactic, with part-of-speech tags
vague or ambiguous adverbs	AH	lexical, semantic and pragmatic	syntactic, with part-of-speech taggs

from the Ambiguity Handbook, so we had the total of 27+20-9=38 patterns included in the tool. The detection patterns that were finally implemented in the tool are presented in Table 2. To make the table compact, we merged similar patterns to a single line of the table, so there is no 1:1 correspondence between table lines and patterns. Here, it is important to emphasize that many of the ambiguity detection patterns implemented in our tool represent semantic or pragmatic ambiguities, although we perform solely lexical and syntactic analysis. This is also easy to see in Table 2.

4 Ambiguity Detection and Presentation

Technically, our ambiguity detection tool is basically similar to the Unix tool `grep`: it investigates the input text line by line, checks whether the analyzed line matches certain regular expressions, and, in the case of matching, marks the found match (ambiguity) in the analyzed line. The tool implementation goes beyond pure `grep` by modularizing the definitions of ambiguity types and regular expressions used to detect these ambiguity types. This allows for tool extensions even by people not familiar with the tool architecture. Tables 3-5 present the keywords and regular expressions used for ambiguity detection. For this presentation, we use standard notation for regular expressions, with "." denoting any character, "|" denoting set union, and "*/+" denoting iteration.

The ambiguities are grouped by the detection techniques:

- Table 3 (lexical patterns) presents the ambiguities resulting from the fact that certain words always entail several interpretations. Such ambiguities can be detected on the lexical level, so Table 3 presents single keywords used for ambiguity detection. The only constraint when searching for these keywords is that we have to search for whole words only, and must ignore keywords occurring as constituents of longer words. Otherwise, the tool would match "*or*" with "m*or*e". Thus, the tool implements special measures for whole word matchings.
- Table 4 (word combinations) presents ambiguities resulting either from co-occurrence of certain words or from occurrence of certain word in special positions. As for Table 3, the tool matches whole words or phrases only.
- Table 5 (syntactic patterns) presents three ambiguities whose detection requires part-of-speech tagging: passive voice and usage of adjectives and adverbs. As the output of the TreeTagger consists of triples (e.g. `is°VB°be`), the regular expressions are rather complex, since they have to cover complete triples to ensure a correct presentation of the error messages.

  Passive voice: To detect passive voice, we proceed as follows: the basic idea is to search for the verb "to be", followed by the past participle form. This search pattern can be expressed in the regular expression "be.*VBN", where the tag `VBN` denotes the participle form. Explicit search for different forms of the verb "to be" is unnecessary, as the applied POS-Tagger provides not only the tag, but also the basic form for every word. The problem is, however, that the above regular expression matches too much in compound sentences. For example, in the sentence "it *is* an interesting idea that can *be* further *explored*", it matches the whole text piece from "is" to "explored". In order to make the match more precise, we refined the detection rule to the following: passive is detected by

occurrence of the verb “to be”, followed by the past participle, but no further verbs are allowed to occur between “be” and the participle. Such a word sequence can be matched by the regular expression presented in Table 5.

Adjectives and adverbs: The list of vague adjectives and adverbs in the Ambiguity Handbook is incomplete, as it contains “etc”. When analyzing manual evaluations (cf. Section 5) we came to the conclusion that there are a lot more adjectives and adverbs that are perceived as ambiguous, than listed in the Ambiguity Handbook. So, we decided to trade in some precision for recall and to mark every adjective and every adverb as a potential ambiguity. Marking of adjectives as a potential ambiguity source is also in line with the statement by Rupp [12] that any adjective in comparative form (“better”, “faster”) can result in misinterpretations.

Tables 3-5 clearly show that most ambiguities result from single ambiguous words (not from word combinations) and, thus, can be detected on the lexical level. To apply the tool to German documents, we use the same regular expressions, with the only difference that the keywords are translated and the regular expression for passive detection is altered to fit German grammar.

The tool marks every found ambiguity occurrence either red or orange or blue, depending on the severity of the found ambiguity. This marking idea is similar to errors and warnings produced by most compilers: An ambiguity is marked red, if it definitely represents a problem, and either orange or blue, if it, depending on the context, can be

Table 3. Keywords used to detect ambiguities on the lexical level

until, during, through, after, at	These expressions do not specify the “outside” behaviour.	Maintenance shall be performed on sundays vs. only on sundays.
could, should, might	These expressions are not concise.	The system should avoid errors.
usually, normally	Unnecessary speculation	The system should not display errors normally.
actually	Requirements shall avoid possibilities	Actually, this requirement is important.
100 percent, all errors	Wishful thinking	The system must be 100 percent secure.
he, she, it	Potentially unclear reference.	The system uses encryption. It must be reusable.
(,)	Unclear brackets	The system shall use HTML (DOC) documents.
/	Unclear slashes	The System shall use HTML/DOC documents.
tbd, etc	These expressions denote that something is missing	The system shall support HTML, DOC etc.
fast	Vague non functional requirement	The system shall be fast.
this	Potentially unclear reference.	This is very important.

Table 3. (*Continued*)

until, during, through, after, at	These expressions do not specify the "outside" behaviour.	Maintenance shall be performed on sundays vs. only on sundays.
could, should, might	These expressions are not concise.	The system should avoid errors.
usually, normally	Unnecessary speculation	The system should not display errors normally.
actually	Requirements shall avoid possibilities	Actually, this requirement is important.
100 percent, all errors	Wishful thinking	The system must be 100 percent secure.
he, she, it	Potentially unclear reference.	The system uses encryption. It must be reusable.
(,)	Unclear brackets	The system shall use HTML (DOC) documents.
/	Unclear slashes	The System shall use HTML/DOC documents.
tbd, etc	These expressions denote that something is missing	The system shall support HTML, DOC etc.
fast	Vague non functional requirement	The system shall be fast.
this	Potentially unclear reference.	This is very important.

Table 4. Regular expressions used to detect ambiguities resulting from word combinations

Regular expression	Matched ambiguity	Example
up to (?!including|excluding)	Up to with unclear inclusion	The system shall support up to five concurrent users.
everybody .* their, everybody .* its	Either language error or ambiguous plural	Everybody uses their login id.
^both	At the beginning of the sentence, both has a unclear reference	Both should be documented.
(all|each|every) .* (a|his|her|its|their|they)	Dangerous plural with ambiguous reference	Every student thinks she is a genius.
and .* or, or .* and	The combination of "and" and "or" leads to unclear associativity	The system shall read HTML and PDF or DOC files.

potentially unambiguous, too, cf. Table 3. The difference between blue and orange is based on our experiments with the tool: patterns that get blue markings are more likely to result in false positives. For every sentence that contains a detected ambiguity, the tool places a pictogram next to the sentence. If the user clicks on the pictogram, he/she will get an explanation for every marking in the sentence under analysis. Additionally, the tool user gets a short explanation if he/she places the mouse pointer over the marked text. Figure 1 shows an example of the presentation of found ambiguities to the user.

Table 5. Regular expressions involving Part-of-Speech tags

Regular expression	Matched ambiguity	Example
\b\w+?°V[^°]*°be (\W[^°]+?°(?!VB.)[^°]*°[^]+?)* \W\w+?°VBN°\w+	Authors should state requirements in active form, as passive conceals who is responsible for the action.	The system will be tested.
\w+?°JJ.?°[^]+	All adjectives.	The system shall be fast and configurable.
\w+?°RB.?°[^]+	All adverbs.	The system shall save data permanently.

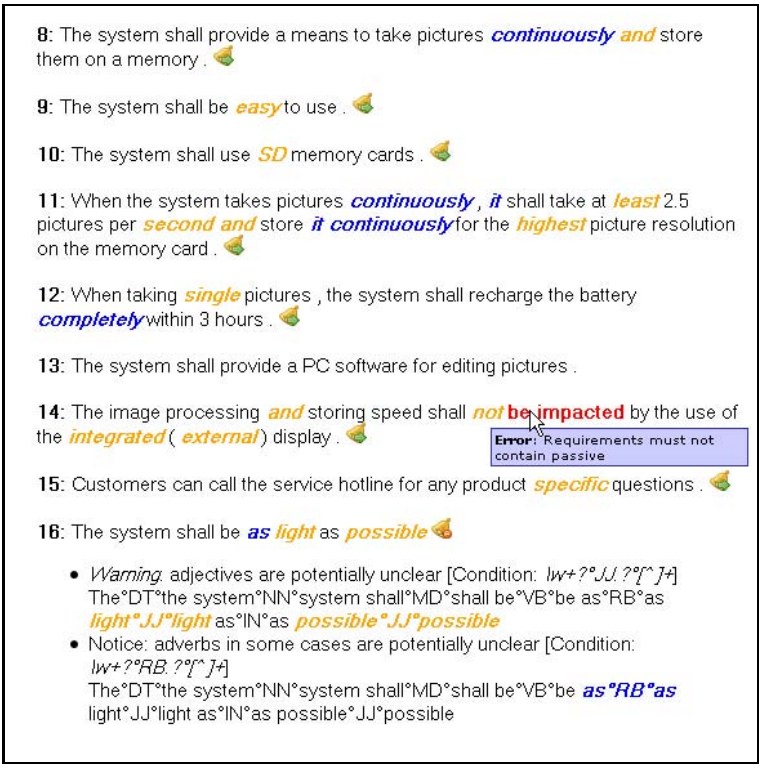


Fig. 1. Sample tool output, applied to the text used for evaluation (cf. Section 5)

5 Evaluation

In order to evaluate the tool, we created a reference data set consisting of approximately 50 German and 50 English sentences. The sentences were not specially crafted for the tool evaluation, but taken at random from real requirements documents. Table 6 shows an excerpt from the reference data set.

We asked 11 subjects to mark ambiguities in the data set. We had subjects from different backgrounds:

Table 6. Reference data set, excerpt

1. The system should be easy to use.
2. The system shall have a world class industrial design.
3. The system shall be as light as possible.

- 5 full-time requirements engineers and software consultants at Siemens,
- 2 software engineering master students at the University of Augsburg,
- 3 PhD students and 1 postdoc at the Technische Universität München, all doing research in requirements engineering.

The subjects were not trained in ambiguity detection, nor were they provided with theoretical backgrounds like the Ambiguity Handbook. Our subjects were given the following instructions[1]:

> You should find deficiencies in the provided text, especially ambiguities or imprecise passages, that negatively influence the requirements.
>
> The found errors can be marked on paper or submitted in some other way. It is important just to mark the text passage that you find problematic and to estimate the criticality of the passage, on the scale from 1=very little relevance to 10=highly crucial.
>
> The report on text deficiencies can have, the following form, for example:
>
> ... this should be actually done by the system ...
> Found problem: fuzzy phrasing; criticality: 3

Table 7 shows the same excerpt form the reference data set as Table 6, with markings by our subjects. It is easy to see that the same ambiguities were marked in different ways by different subjects: Subject 1 found the phrases from Table 6 unambiguous, Subject 2 marked single words or short phrases, and Subject 3 mostly marked extensive phrases. Furthermore, Subjects 2 and 3 attached different criticality values to the same ambiguities.

We considered an ambiguity as present in the evaluation text if it was marked by at least one subject, and defined the extents of the ambiguous phrase as the longest continuous word sequence marked by some subject. This means that we considered "easy to use" as the ambiguous phrase in Sentence 1 from Table 6. To define a criticality of a given ambiguity, we assigned the average criticality value provided by our subjects.

The rationale for this decision was that we wanted to evaluate the reliability of the tool. More precisely, we wanted to evaluate whether manual analysis still remains necessary after tool application. In the case that the tool is absolutely reliable, the human analyst would solely have to decide whether potential ambiguities found by the tool are genuine ambiguities, but he/she would not have to search for other ambiguities manually. Thus, as a first approximation, it was necessary to treat *every* ambiguity marked by *some* subject as a genuine ambiguity.

To evaluate the tool performance, we used the following definitions: Let E be the set of ambiguities found by human evaluators, T be the set of ambiguities detected

[1] The original instructions were in German, here we provide an English translation

Table 7. Reference data set with markings by our subjects, excerpt

Subject	Markings	Criticality
Subject 1	1. The system should be easy to use.	—
	2. The system shall have a world class industrial design.	—
	3. The system shall be as light as possible.	—
Subject 2	1. The system should be easy to use.	7
	2. The system shall have a world class industrial design.	7
	3. The system shall be as light as possible.	7
Subject 3	1. The system should be easy to use .	10
	2. The system shall have a world class industrial design .	10
	3. The system shall be as light as possible .	8

by the tool, and $S = T \cap E$. In the most simple form, we calculated recall and precision as $Precision = \frac{|S|}{|T|}$ and $Recall = \frac{|S|}{|E|}$. Additionally, we used the criticality values to calculate weighted recall. We defined $Recall_{weighted} = \frac{weight(S)}{weight(E)}$, where $weight(A) \stackrel{def}{=} \sum_{a \in A} criticality(a)$. Weighted precision makes no sense, as we would have to mix unrelated criticality values coming from different sources.

When calculating recall and precision according to the above definitions, we observed two interesting phenomena:

1. There exist text passages that were marked as problematic, but these markings represent no ambiguities but are purely stylistic. Furthermore, they are contradictory to explicitly stated best practices by Siemens or the suggestions of the Ambiguity Handbook. For example, one of our subjects marked "shall" as ambiguous, although the use of "shall" is not ambiguous at all and is even an explicitly stated best practice by Siemens. On the other hand, another subject marked every requirement that was in indicative mode, which is more a matter of taste than a real ambiguity. In the following definitions we will refer to such ambiguities as "$BP-$" (falsely marked ambiguities that are not ambiguities according to best practices). Here, it is important to emphasize that *no* ambiguity was absorbed into $BP-$ simply because it was not contained in the Ambiguity Handbook or Siemens guidelines. Marked ambiguities were absorbed into $BP-$ only if they were purely stylistic and in contradiction with best writing practices.
2. There exist ambiguities that were missed by every subject (thus, these ambiguities were not in E), which are still genuine ambiguities in the sense of the Ambiguity Handbook or the Siemens-guidelines. For example, many occurrences of passive voice were missed by the subjects. In the following definitions we will refer to such ambiguities as "$BP+$" (genuine ambiguities according to best practices, which were not found by our subjects).

In order to attenuate the influence of human evaluators' performance on the tool evaluation, we evaluated the tool not only with the original set of marked ambiguities (E), but also with $E \cup BP+$, $E \setminus BP-$ and $(E \cup BP+) \setminus BP-$ as reference sets. As the sets $BP+$

Table 8. Evaluation results

Language	Reference set	Precision (%)	Recall, simple (%)	Recall, weighted (%)
English	E	47	55	64
	$E \cup BP+$	95	71	78
	$E \backslash BP-$	95	75	77
	$(E \cup BP+) \backslash BP-$	**95**	**86**	**86**
German	E	34	53	52
	$E \cup BP+$	97	76	74
	$E \backslash BP-$	97	69	70
	$(E \cup BP+) \backslash BP-$	**97**	**86**	**86**

and $BP-$ are disjoint, it makes no sense to evaluate $(E \backslash BP-) \cup BP+$ separately, as it coincides with $(E \cup BP+) \backslash BP-$. Evaluation results are presented in Table 8.

For an ambiguity detection tool, the recall value is definitely more important than precision. In the ideal case, the recall should be 100%, as it would allow to relieve human analysts from the clerical part of document analysis [4]. For our tool, while the raw recall value of about 50% is rather low, the revised values ($(E \cup BP+) \backslash BP-$) of 86% show that the tool is fit for practical use, as it marks six of seven errors detected by humans and consistent with best practices. The precision is high, especially when $BP+$ is taken into account. Not every error detected by our tool is also detected by humans, but nearly every (95-97%) error found by the tool is based on a best practice from literature.

As for the errors marked by human evaluators but missed by the tool, manual analysis has shown that they represent either language errors, or pragmatic ambiguities where we could not identify explicit lexical or part-of-speech patterns that would allow to automate error detection. Language errors were presented by missing subject in two sentences: in "It assumed that for image call-up…" and in "Wants to have picture parameters…" A test with the C&C parser [13] has shown that one of these errors can be detected due to the incompleteness of the resulting parse (not yielding a complete parse tree), whereas the other would require a more thorough analysis of the resulting parse tree. Due to the computational complexity of the required analysis, C&C is rather slow, especially for interactive applications, when compared with the TreeTagger. Because of this and a rather small gain (at most two new errors from the reference data set would become detectable), we decided not to extend our tool in order to include parsing.

Pragmatic ambiguities not detected by the tool are more subtle: they occur in perfectly sensible and grammatically correct sentences, and profound knowledge of the application domain is necessary to spot the ambiguity. For example, our subjects marked ambiguities in following sentences:

- "If archiving of data failed even after a reasonable number of retries, the system shall store data locally and inform the user that it is currently not possible to archive data." (Evaluator's comment: what is the difference between archiving and storing?)
- "It assumed that for image call-up via a native client application installed on the Office PC there is the need to install some imaging components on the Office PC

(primarily parts of the OEM application).” (Evaluator’s comment: “need to install”: who should perform the installation?)
- “The system shall allow taking at least 2 pictures per second.” (Evaluator’s comment: “When does a picture count as taken: when the button is pressed or when the picture is stored?”)

Unfortunately, detection of such ambiguities is far beyond the capabilities of the state-of-the-art computational linguistics.

In addition to calculating the recall and precision of the tool, we calculated the recall of the human evaluators, taking E (set of ambiguities marked by at least one human evaluator) as the reference set. Surprisingly, it turned out that the average recall of a human analyst was at just 19%. Many of our subjects admitted, when submitting evaluation results, that attentiveness rapidly decreases when reading as little as 3 pages and that their markings were most probably influenced by this effect. However, a χ^2 test did not support the hypothesis that evaluators’ performance decreased during reading of the provided text. Nevertheless, due the *perceived* performance decrease, an application of automated ambiguity detection can be helpful. Furthermore, although our tool is not perfect (recall below 100%), it detects a lot more genuine ambiguities than an average human analyst. Due to the large number of ambiguities that were not perceived by human analysts as such, it would be an interesting direction for future research, to investigate if a completely disambiguated text would still be perceived as a natural text.

6 Related Work

Lightweight text processing techniques (techniques not involving natural language parsing) are very popular in requirements analysis, as they are easy to implement and, nevertheless, can provide valuable information about document content. Such techniques can be used, for example, to identify application specific concepts. Approaches by Goldin and Berry [14], Maarek and Berry [15], and Sawyer et al. [16] provide good examples of concept extraction techniques: they analyze occurrences of different terms, and basing on occurrence frequency, extract application-specific terms from requirements documents. The focuses of these approaches and our approach are different, though: we do not perform any concept extraction but focus exclusively on ambiguity detection.

Ambiguity detection approaches are closer to the presented tool and should be analyzed more thoroughly. Apart from the approaches by Berry et al. [1] and Kiyavitskaya et al. [4], used as the basis for the presented tool, ambiguity detection approaches were introduced by Fabbrini et al. [17], Kamsties et al. [18], and Chantree et al. [19]. The approach by Fabbrini et al. introduces a list of weak words and evaluates requirements documents on the basis of weak word presence. Weak word detection is already included in our tool, and adding further weak words to the detection engine is just a matter of extending the weak words database. The ambiguity types classified by Kamsties et al. became a part of the Ambiguity Handbook later, so our tool already covers most ambiguities presented there. The approach by Chantree et al. deals exclusively with the coordination ambiguity. Our tool, although not specially designed to detect coordination ambiguity, is able to detect coordination ambiguity, too, and, in addition to that a lot more other types of ambiguities.

The tool presented in this paper has one important advantage when compared to other existing ambiguity detection approaches: It can not only detect ambiguities, but also explain the rationale for the detected ambiguity. Thus, apart from pure ambiguity detection, the presented tool can be used to educate requirements analysts, too.

7 Summary

Ambiguous requirements introduce conflict potential to a software project, as different stakeholders can interpret them in different ways, and then argue, whose interpretation is the correct one. One way to avoid such problems is to detect ambiguities early in requirements analysis. The presented tool, although performing lexical and syntactic analysis only, is able to detect ambiguities on all levels, from lexical to pragmatic. Although not able yet to detect all ambiguities, as listed in the Ambiguity Handbook, the tool represents an important milestone in the development of a tool completely satisfying the requirements to ambiguity detection tools by Kiyavitskaya et al. [4]: The presented tool is able not only to detect ambiguities, but also to provide explanations for detected ambiguities. This makes the presented tool suitable not only for ambiguity detection, but also for education purposes. In the industrial requirements engineering, these benefits have the potential for considerable time and cost savings while enabling a higher quality of requirements at the same time. The modular and lightweight design of our tool facilitates integration and customization for many practical applications.

Acknowledgments

We want to thank the participants of our empirical evaluation and other people who helped to improve this paper: Bernhard Bauer, Naoufel Boulila, Andreas Budde, Roland Eckl, Dominik Grusemann, Christian Leuxner, Klaus Lochmann, Asa MacWilliams Daria Malaguti, Birgit Penzenstadler, and Carmen Seyfried.

References

1. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity (2003), `http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf` (accessed 27.12.2009)
2. Mich, L., Franch, M., Novi Inverardi, P.: Market research on requirements analysis using linguistic tools. Requirements Engineering 9, 40–56 (2004)
3. Kamsties, E., Knethen, A.V., Philipps, J., Schätz, B.: An empirical investigation of the defect detection capabilities of requirements specification languages. In: Proceedings of the Sixth CAiSE/IFIP8.1 International Workshop on Evaluation of Modelling Methods in Systems Analysis and Design (EMMSAD 2001), pp. 125–136 (2001)
4. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.M.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requir. Eng. 13, 207–239 (2008)

5. Santorini, B.: Part-of-speech tagging guidelines for the Penn Treebank Project. Technical report, Department of Computer and Information Science, University of Pennsylvania (3rd revision, 2nd printing) (1990)
6. Schiller, A., Teufel, S., Stöckert, C., Thielen, C.: Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Institut fur maschinelle Sprachverarbeitung, Stuttgart (1999)
7. Schmid, H.: Probabilistic part-of-speech tagging using decision trees. In: Proceedings of the International Conference on New Methods in Language Processing, pp. 44–49 (1994)
8. Schmid, H.: Improvements in part-of-speech tagging with an application to german. In: Proceedings of the ACL SIGDAT-Workshop, pp. 47–50 (1995)
9. Russell, S., Norvig, P.: Communicating, perceiving, and acting. In: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
10. Kof, L.: On the identification of goals in stakeholders' dialogs. In: Paech, B., Martell, C. (eds.) Monterey Workshop 2007. LNCS, vol. 5320, pp. 161–181. Springer, Heidelberg (2008)
11. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich (1999)
12. Rupp, C.: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis, 2nd edn. Hanser–Verlag (2002), ISBN 3-446-21960-9
13. Clark, S., Curran, J.R.: Parsing the WSJ using CCG and log-linear models. In: ACL 2004: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics, Morristown, NJ, USA, p. 103. Association for Computational Linguistics (2004)
14. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. 4, 375–412 (1997)
15. Maarek, Y.S., Berry, D.M.: The use of lexical affinities in requirements extraction. In: Proceedings of the 5th International Workshop on Software Specification and Design, pp. 196–202. ACM Press, New York (1989)
16. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. Softw. Eng. 31, 969–981 (2005)
17. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In: 26th Annual NASA Goddard Software Engineering Workshop, Greenbelt, Maryland, pp. 97–105. IEEE Computer Society, Los Alamitos (2001)
18. Kamsties, E., Berry, D.M., Paech, B.: Detecting ambiguities in requirements documents using inspections. In: Workshop on Inspections in Software Engineering, Paris, France, pp. 68–80 (2001)
19. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: RE 2006: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), Washington, DC, USA, pp. 56–65. IEEE Computer Society, Los Alamitos (2006)

Ambiguity in Natural Language Software Requirements: A Case Study

Fabian de Bruijn and Hans L. Dekkers

University of Amsterdam,The Netherlands
eb i h e e

Abstract. [Context and motivation] Ambiguous requirements are often seen as a cause for project failure, however there is little empirical data to support this claim. **[Question/problem]** In this research we study the effect of a highly ambiguous requirements document on project success. **[Principal ideas/results]** The studied project was a complex data processing system that took about 21 man year to develop. First, we determined the level of ambiguity by three independent tests. Next, we did a root cause analysis on a selection of the main issues to establish if ambiguous requirements were a significant cause. Surprisingly, this case study shows that only one of the examined failures was caused by ambiguous requirements. Both the independent test team and the third party development team found ways to cope with the high level of ambiguity. For the development team this required a substantial investment to clarify requirements. **[Contribution]** The main contributions of this paper are the counterintuitive findings, the collected empirical data and the method used to collect these data.

Keywords: Requirements specification, Ambiguity, Natural language, Empirical.

1 Introduction

Requirement specifications serve as contract, a starting point for development, and a focal point for quality control. It is generally considered to be vital that requirements are unambiguous [2,4,9,12,15]. Formal languages serve this purpose, however, since formal languages are not well understood by most stakeholders, natural language requirements are the de facto standard. Berry *et al.*[4] make a strong case for writing unambiguous natural language specifications and provide a handbook which describes a taxonomy of different types of ambiguity and how to avoid them. Since natural language is inherently ambiguous [3], avoiding ambiguity is by no means a trivial task.

We wonder just how important it is to minimize ambiguity and how much effort should be invested. The agile movement puts the focus on communication and feedback. They stress that writing unambiguous requirements is an illusion[16] and that even if requirements are unambiguous, problems of validity, volatility, and correct interpretation remain. Still, the general practice of tenders and contracts for outsourced projects demands a requirement specification. In this research we study the effect of a highly ambiguous requirements document on project success.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 233–247, 2010.

1.1 Research Question

Our main research question is:

> *What is the effect of ambiguity in the requirement specification on project success?*

The initial step we have taken is to analyze a real life project:

1. *How many requirement statements are ambiguous?*
2. *How many problems were caused by ambiguous requirements?*

2 The Importance of Unambiguous Requirements

2.1 Communication in Requirements Engineering

Requirements engineering is the process in which stakeholder needs are elicited, gathered, analyzed and in which decisions are made on the requirement set for the product to be built. Requirements are an abstraction and a perception of the true needs of the stakeholders. It is the result of a creative process where communication is crucial[17]. It is not evident that the requirement set as a whole is feasible, complete or correct.

In the context of this paper it is important to distinguish between the stakeholder need, the requirement statement and the way it is understood[20]. We study the effect of ambiguous requirement statements on the understanding of the developers. This understanding also depends on context information (goals, rationale, domain description), domain knowledge and personal factors.

Requirement specifications are not standardized and many different types of requirements exist [1,12,17,15]. It is good practice to write requirements in the problem domain, leaving the design space open for the development team. This poses the interesting problems of how to be specific and how to determine if a solution satisfies the requirements. To illustrate this consider usability requirements. Usability can be measured by the number of tick and clicks and user errors. But how to set a norm: what is an optimal solution, when is a suboptimal solution still acceptable?

2.2 Related Work

In software engineering literature there is no single definition of ambiguity. Several authors have given different interpretations and different causes for ambiguity. For instance, Davis[8] states that when a requirement can be interpreted in two or more ways then this requirement is ambiguous. Schneider *et al.*[19] mention that ambiguity is caused by an essential part in a software requirement that has been left undefined or defined in a way that causes confusion among humans. Berry *et al.*[4] focus on ambiguities which are caused by expression inadequacies.

There is general consensus that requirements should be unambiguous or at least that ambiguity should be recognized and intended. Much work has been done to achieve quality in requirements [2,12,15,18] and reduce the ambiguity in natural language software requirements. In [4,13,14] methods and rules are discussed to surface ambiguity in natural language text including techniques like checklist and scenario based reading.

Fabbrini et al. [10,9] present a tool to analyze the quality of natural language requirements written in English. The tool Alpino[5] can be used to automatically parse texts written in Dutch, to find the most probable interpretation.

3 Research Method

To answer the research question "What is the effect of ambiguity of software requirements on project success?" we studied a real life project that failed. First we established the level of ambiguity in the requirements specification. Next we established if ambiguity was a significant cause for the reported issues.

3.1 Case Study Project Information

The project started in December 2007 and was canceled in May 2009 after acceptance tests by an independent test team found over 100 blocking issues. The contractor was convinced that ambiguity was an important cause of many of the reported issues. The case study is referred to as Project X. For reasons of confidentiality we cannot fully disclose requirements. Project X was the development of an Oracle system that was composed of a GIS component, a data processing component and a rule engine connected by an enterprise service bus. Two of these components were existing systems that were adapted for this project. Project X took well over 40.000 man hours, roughly 21 man year; a significant budget overrun given the initial estimation of 10.000 hours. 80 persons worked on the project.

The requirement specification was in Dutch. From the start the requirements were considered to be unclear. To get more clarity workshops were organized and an elaborate design was made, however, when asked, the customer did not formally agree with the design. The independent test team based its findings on the initial requirements document.

3.2 Establishing Ambiguity

We consider a requirement to be ambiguous when it has at least two different valid interpretations. To establish ambiguity three independent tests were performed on a sample set of the requirements:

- three professionals searched for differences in interpretations;
- a systematic review by two software engineers based on the taxonomy of [4];
- an automatic analysis by natural language analysis tool Alpino [5] .

3.2.1 Sample Data

The sample size was set to 102[1] from a total of 279 requirements. A stratified random sample [6] was chosen from the requirement specification. All tests to detect ambiguity

[1] After the first day of reviewing a sample of 100 requirements was considered feasible. To get an even distribution over the requirement categories we took the % and rounded this. E.g. if a requirements category consisted of 5.6% of the complete number of requirements, we took 6 requirements from this set. This resulted in a complete sample of 102.

were performed on this sample. Table 1 lists the different requirements categories, % of total is the total number of requirements in the category / total no. of requirements.

3.2.2 Review Panel Interpretations

The requirement statements were reviewed by two requirement engineers and one software engineer. The reviewers were experienced professionals with no prior knowledge of Project X and with a similar domain knowledge as the development team. During $2\frac{1}{2}$ days they studied each requirement and wrote down the interpretation they thought was most fit. To determine if interpretations differed, a joint session was organized by the first author, taking half a day. Reviewers shared their interpretations and could judge if they had the same understanding. To make sure that interpretations were plausible and truly different the reviewers had to present some form of argument or example.

Table 1. Requirement categories

ID	Category	% of total	sample size
1	Functionality A	19,00%	19
2	Functionality B	13,98%	14
3	Functionality C	8,96%	9
4	Functionality D	8,60%	9
5	Functionality E	7,53%	8
6	Infrastructure	6,81%	7
7	Maintainability	5,02%	5
8	Software features	5,02%	5
9	Functionality F	4,66%	5
10	Usability	4,30%	4
11	Security	3,94%	4
12	Data model	3,94%	4
13	Reliability	3,23%	3
14	Performance	2,51%	3
15	Functionality G	2,51%	3

3.2.3 Systematic Review

The review was carried out by the first author and a software engineer with no prior knowledge of Project X. A checklist was used, based on the linguistic ambiguity taxonomy of Berry *et al.*[4]. The reviewers had no linguistic background. The review protocol was as follows.

1. Each reviewer individually determines the ambiguity types present in a requirement statement.
2. The results are merged, showing the identified ambiguity types.
3. Each reviewer can adjust his findings based on the merged results.
4. Finally consensus was reached in a face to face meeting.

3.2.4 Automatic Analysis

To get an objective measure for ambiguity each requirement was parsed by the tool Alpino[2][5]. Alpino recognizes lexical and structural (or syntactical) ambiguity. The output showed the possible parse trees. The number of possible parses was used as indicator for ambiguity. Not all of these parses have a clear semantics or are semantically different. The limit of parses was set to a maximum of 50.

[2] Alpino version 14888 for the x86 Linux platform was used in this experiment.

3.3 Relating Issues to Ambiguous Requirements

To establish if ambiguity was a significant cause for project failure, we did a systematic root cause analysis on a sample of issues found by an independent test team. The issue tracking system used by Project X was JIRA. Issues have fields to describe and reproduce the issue (e.g. description, version, component); to capture the status of the issue (e.g. type, status, priority) and a chronological log showing the comments on the issues and steps to clarify and solve these by the developers. Sometimes attachments were presented containing snapshots of the system, or a specification of desired behavior by the test team.

Table 2. Root cause conditions

Label	Root cause condition
RCC1	The implementation satisfies the requirement.
RCC2	The test team rejects this because of a different but also valid interpretation of the requirement.
RCC3	Test team respects the design space of the contractor[3].
RCC4	Disambiguation of the requirement would have prevented these differences in interpretation

In total 256 issues were reported, we focused on the 125 issues that were considered blocking or urgent. 16 of these were addressing problems like incomplete product documentation and were excluded from our research. The 109 remaining issues were related to the requirement categories. While reading the issues to relate them to requirement categories, 5 issues surfaced that were likely to have been caused by ambiguity. These issues were either labeled as such or there was a discussion about the interpretation. A root cause analysis was performed on these five issues and on a random selection of 35 issues. The issues were annotated by discussion threads of the contractor which were also studied. All issues were analyzed by each of the reviewers individually.

We consider ambiguity to be the cause of an issue if all conditions in table 2 are satisfied.

4 Results

4.1 Measured Ambiguity

4.1.1 Review Panel Interpretations

The review panel of three different persons qualified 36 requirements as unambiguous, 41 requirements had two different interpretations, and finally 25 requirements had three different interpretations. Figure 1 shows for each requirement category how many requirements were ambiguous and how many were unambiguous.

[3] Many requirements are formulated in the problem domain, explicitly leaving the choice of solution to the contractor. In some cases the test team prefers a solution different from the one that is implemented while it is apparent that the requirement is interpreted in the same way as the contractor. These issues are not considered to be caused by ambiguity.

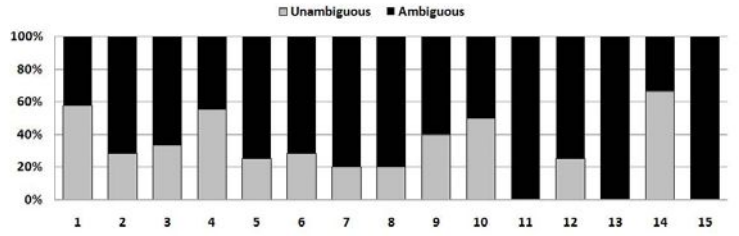


Fig. 1. x-axis: requirement categories, see table1. y-axis: % of requirements.

Table 3. Ambiguity types top 5 found in requirements sample

Ambiguity type	# Requirements
Vagueness	58
Language error	32
Coordination Ambiguity	11
Scope Ambiguity	9
Attachment Ambiguity	8

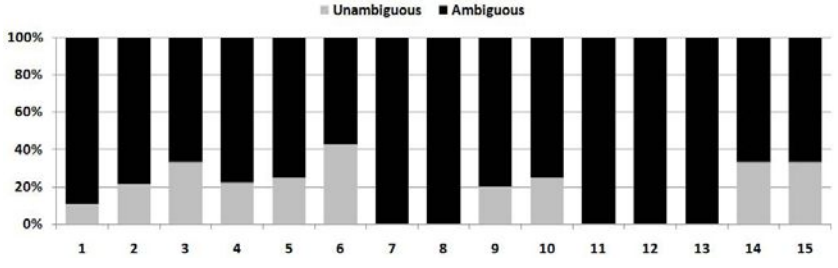


Fig. 2. x-axis: requirement categories, see table1. y-axis: % of requirements.

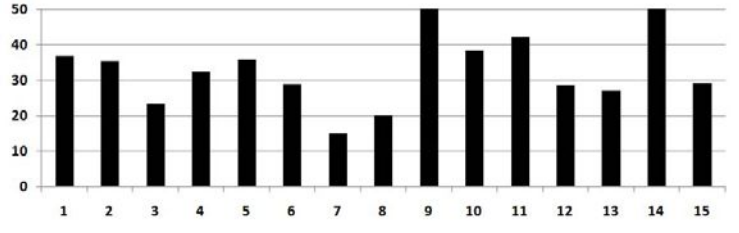


Fig. 3. x-axis: requirements categories, see table 1. y-axis: average number of parse trees per requirement.

4.1.2 Systematic Review

The systematic review found 20 of the requirements to be unambiguous. 82 requirements were found to be ambiguous. In 36 of the requirements multiple types of ambiguity were

discovered. Table 3 lists in how many requirement statements the most common ambiguity types were found. Note that the number of ambiguities within one single requirement statement is not counted. Figure 2 shows for each requirement category how many requirements were ambiguous and how many were unambiguous.

4.1.3 Alpino Tool Interpretations

For 94 of the requirements Alpino found more than one parse tree, meaning that these are syntactical ambiguous. Seven requirements could not be parsed. Alpino calculated an average of 33 different parse trees for each requirement[4]. The median was 47 and the standard deviation was 19. Only one requirement had one parse tree. This requirement was also understood in one way by the review panel members. Figure 3 shows the average number of interpretations for each requirement category.

4.2 Issue Causes

40 issues were analyzed. The root cause of these issues is presented in table 4. We found one issue to be caused by ambiguity. The issue was "Processing batch file takes too long". The corresponding ambiguous requirement was "The processing of the batch programs should be completed within a reasonable time frame without affecting the performance of the application and therefore without impeding the usual business." For the contractor it was and still is not clear when the processing time is acceptable. The performance was improved, the effort is discussed in section 5.

Table 4. Results issue root cause analysis

Root cause	No. of issues	Explanation
Ambiguity	1	Issue caused by ambiguity
Feature not found	3	Feature had been implemented, but the implementation was not known to the test team.
Missing requirement	5	For these issues no requirement could be identified.
New feature request	1	The reported issue is not required by the current requirements.
Incorrect implementation	27	Acknowledgment by contractor that the issue reports undesired behavior
Not reproducible	3	These issues could not be reproduced
Total no. issues	*40*	From 109 blocking or urgent issues

To make our reasoning process transparent we illustrate this in table 5. The presented issue was selected because it raised some discussion and gives good insight in our analysis. Determining if the implementation satisfied the requirements was sometimes straightforward, sometimes it was hard to ascertain. The issue and requirements were in Dutch and there is some risk that ambiguity is lost in translation. Also the issue description and annotations are too large to be presented.

[4] Calculations over the set of requirements that could be parsed.

Table 5. Reasoning process details: issue I23

(a) Context information

Issue caption	The application can not be controlled by the keyboard. Note: the test team provides some examples of expected key controls.
Requirement	The whole system has a consistent user interaction. The application can both be controlled by keyboard and by mouse, based on a completely WEB oriented graphical user interface.

(b) Reasoning process

RCC1	TRUE	A consistent keyboard interface has been implemented. To use the application the mouse is not required. The development team agrees that the keyboard control can be improved upon, however this is considered to be a new feature request.
RCC2	TRUE	The test team provides some examples of keyboard controls that they had expected to be implemented but were not.
RCC3	FALSE	The comments and examples of the test team show that they interpret the requirement identically to the development team. From the issue text it becomes clear that the test team is not knowledgeable of the keyboard controls that have been implemented. There is no indication if they find the current implementation acceptable.
RCC4	FALSE	The requirement is specific enough to judge the present implementation. The vagueness in the requirement leads to misunderstanding. However this vagueness is intended, so the contractor can choose the specific solution.
Conclusion		Feature not found. For the test team it was not apparent how this requirement was implemented. As they did not find this feature, they filed an issue report. The issue specification is not required by the current set of requirements.

5 Evaluation

Research question: How many requirement statements are ambiguous?

Table 6 shows that the studied requirements sample of 102 requirements revealed a lot of ambiguous requirements. Alpino considered all but 1 requirement to be ambiguous. The systematic review considered 83 out of 102 requirements to be ambiguous. The review panel considered 66 of the 102 requirements to be ambiguous. That the review panel has a single reading for an ambiguous statement corresponds with the notion of innocuous ambiguity[7].

Research question: How many problems were caused by ambiguous requirements?

Only one of the forty inspected issues was caused by ambiguity in the requirements. This issue was not a costly one. From our study we cannot conclude that ambiguous requirements caused the failure of this project.

How come there weren't more issues caused by ambiguous requirements?

Table 6. Ambiguity of requirements according to review panel, systematic review, and Alpino

Review panel	Systematic review	Alpino	Requirements (#)
Unambiguous	Ambiguous	Unambiguous	1
Unambiguous	Unambiguous	Ambiguous	9
Unambiguous	Ambiguous	Ambiguous	27
Ambiguous	Unambiguous	Ambiguous	10
Ambiguous	Ambiguous	Ambiguous	48
Ambiguous	Ambiguous	Parse Error	7

Although the requirements specification from project X was highly ambiguous most of the examined issues could not be attributed to ambiguity. Project X used workshops involving the customer to clarify the requirements. Yet, even workshops and discussion don't guarantee a good interpretation. Given the complexity of this project, the many issues, and the lack of contact between test team and development team we were surprised by this finding. This is in line with the observation of [11] that the biggest danger is unconscious disambiguation. The software engineer interprets an ambiguous requirement differently then the customers intention, but is unaware of this. In this project the contractor was from the start aware of the high level of ambiguity.

What is the cost of ambiguous requirements?

The issue that was caused by ambiguity is about performance, potentially a costly issue. However the architecture of the application was set up to process vast amounts of data in reasonable time. To get a reasonable performance took roughly 550 hours[5]. The issue was considered to be resolved however from the issue annotations it was apparent that still much was and is unclear about the real time scenarios (how much data in what time slots) and what performance is considered to be acceptable.

The project suffered a major budget overrun of 30.000 hours. The 550 hours for the issue caused by ambiguity is limited. The workshops to clarify ambiguity were included in the budget (180 hours). The project data does not show what part of the budget overrun is caused by ambiguity.

6 Threats to Validity

6.1 Validity of the Tests to Determine Ambiguous Requirements

Is the sample set of requirements representative?

Throughout the research project we have read and interpreted the complete set of requirements intensively. The requirement sample was characteristic for the whole set of requirements. For the different types of requirements the requirement statements follow a similar pattern and use similar words. We found no occurrences of requirements that were more specific than the ones studied in our sample. Since the different requirement types are in different requirement categories, we consider our sample to be representative for the complete set of requirements.

[5] According to the project manager one software engineer worked on the performance optimization for three months.

6.1.1 Threats to Validity of Ambiguity Tests

6.1.1.1 The Interpretations by the Review Panel. The good thing about this test is that ambiguity that does not lead to misinterpretations will not be reported. However, the reviewers might have an invalid interpretation or a different interpretation from the actual project team or customer. This test is not just an indicator of ambiguity, it also says something about the interpretation process of the reviewers. The final threat to validity is that the review panel is under the impression they have a different interpretation, while they actually share the same interpretation (false positive). This last threat could have been avoided by making the interpreters formalize their interpretation as described in [13].

6.1.1.2 The Systematic Review. Detecting ambiguities by humans is a hard task. The reviewers were no trained linguists and unconscious disambiguation makes it easy to miss ambiguity types. We expect that the number of false negatives is rather high. The ambiguities found complied with the taxonomy of [4] and the test protocol ensured that the found ambiguities were analyzed well. We expect that the number of false positives is rather low.

6.1.1.3 Alpino Scan. Alpino was used to get objective measures for ambiguity. When a requirement has at least two parse trees then the requirement has structural or lexical ambiguity. As described in [7] many of these ambiguities have a single reading by humans and are innocuous. Alpino features a maximum-entropy based disambiguation component to select the best parse for a given sentence. From our discussion with the Alpino research group it became clear that there is not a clear threshold that can be used to automatically determine which of the parse trees is a plausible interpretation. This would have enabled us to automatically distinguish between nocuous and innocuous ambiguity. Also to date Alpino has no feature to report the different types of ambiguity. The parse trees of Alpino are used to detect false negatives.

6.1.2 False Positives of the Ambiguity Tests

6.1.2.1 Unambiguous by Review Panel and Ambiguous in Systematic Review. Since the systematic review showed that 80% of the requirements were ambiguous it is especially interesting to learn about false positives. We did two checks. The first was to assess if the reviewers had a correct understanding of vagueness. This was the ambiguity type that was most discovered (in 58 requirement statements, see table 3). The analysis was done by the second author and a researcher with a linguistic background. Three requirements were analyzed and we could conclude that the classification vague was used according to the taxonomy presented in [4].

False positives are most likely to occur in the requirements where only one ambiguity type was revealed and which the review panel found to be unambiguous. The second author examined five of these requirements to see if the found ambiguity types were according to the taxonomy presented in [4]. Fourteen requirements, see figure 4, met this condition. Eleven of these had ambiguity type "vague", two were of type "language error", one had the type "attachment".

The requirements of type language error contained two plurals but both requirement statements contained no verb. The lack of a verb leaves it to the reader to guess about it. Depending on the verb it could be a language error or it could be a scope ambiguity.

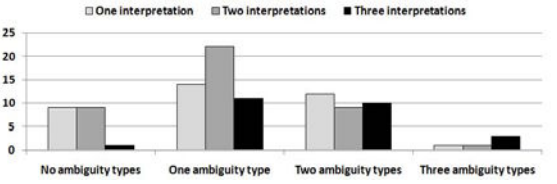


Fig. 4. x-axis: the number of different ambiguity types. y-axis: the number of requirements.

The requirement with ambiguity type attachment was of the form "overview of old and new values with difference percentages". The difference percentages could indeed be attached to new values alone, or to old and new values and to the difference between old and new values. It is also logical that the review panel interpreted this unambiguously as the difference between old and new presented in %. However we would expect that there is still some ambiguity not considered by the review panel. One of the formalizations could be (new value - old value) / old value, now it is easy to see what variations are allowed by this requirement: should we divide by old value or by new value. Should we subtract the old value from the new value or vice versa or take the absolute difference. Clearly an ambiguous statement. Also the two vague requirements were true positives. That the review panel considered these to be unambiguous can be explained that the vagueness was caused by IT terms which were understood identically by the review panel because of their identical IT background.

This analysis revealed no false positives.

6.1.2.2 Ambiguous by Review Panel and Unambiguous in Systematic Review. False positives are also likely to occur in the requirements that were considered to be ambiguous by review panel and unambiguous by the systematic review. Ten requirements met this condition; five of these requirements were analyzed. From the transcripts of the review panel it was apparent that four of these five requirements were understood in a different way and these can be considered to be true positives. In three of these cases the requirement contained an ambiguity type that was unnoticed during the systematic review (these were of types scope and coordination ambiguity). In the fourth case one of the interpreters made a simplification of the requirement that left out important details. This difference in interpretation was not caused by ambiguity.

The fifth case was a possible false positive. This concerned a requirement of the form "Possibility to change the start and end date of a specific process". During the analysis the reviewers were convinced that each of them had a different interpretation. The transcript with their interpretations showed some differences in the sense that the start date can be changed if and only if the end date is changed as well. However it is hard to conceive that the reviewers would have constructed a truly different solution for this requirement.

This analysis revealed one false positive.

6.1.3 False Negatives of the Ambiguity Tests

6.1.3.1 Unambiguous by Review Panel and Unambiguous in Systematic Review. To identify false negatives we analyzed requirements found to be unambiguous by both the review panel and by systematic review. The second author examined these nine occurrences using the checklist of the systematic review. This inspection found two requirements to be ambiguous, both of type attachment. It was easy to see why these were missed the first time; the semantics of the requirement only allowed one sensible interpretation. For four requirements for which the second author also could not identify an ambiguity type the Alpino parses were analyzed. This revealed no new ambiguous requirements. Alpino had multiple parse trees for compound words (finding the right grouping), for words that in Dutch are sometimes used as verb and sometimes as adjective and with the use of ":" as classifier. The different parse trees have the same semantics, and are examples of innocuous ambiguity[7].

This analysis revealed no false negatives as the ambiguous requirements only had one interpretation.

6.2 Validity of the Root Cause Analysis

Is the sample set of issues representative?

A first read of all blocking and urgent issues revealed five issues that were likely to have been caused by ambiguity. Indeed all of these five issues sparked a lot of discussion. Initially we extended this set of five with a random sample of 20 issues. When our first analysis revealed that only 1 issue was caused by ambiguity, another 15 issues were randomly selected and analyzed. The analysis showed that the new set of 15 issues had similar causes as the initial set of 20 randomly selected issues. This strengthens our belief that the sample was representative

6.2.1 Threats to Validity

Since few issues were found to be caused by ambiguity, there is limited danger of false positives. To reduce the number of false negatives the root cause analysis followed a formalized protocol and was carried out autonomously by both authors. The controversial issues have been discussed in depth and clear reasons were found to eliminate ambiguity as root cause.

6.2.2 False Negatives

The category least discussed is the category in which the contractor clearly acknowledges that the implementation is not conform the requirements. This was also the biggest category with 27 of the 40 researched issues. Thirteen of these issues were inspected further. To our surprise for 9 of the 13 cases no requirements were found. The contractor was given a lot of freedom in choosing the design. The issues could be seen as improvements or comments on the design choices made. Even though the

improvements and comments were not specified by the requirements, the contractor felt that the improvements and comments were reasonable and without much discussion qualified each issue as bug. Most issues had been resolved at the time of the research. Our analysis would classify these 9 issues as "new feature request". This analysis of 13 issues revealed no false negatives.

7 Other Observations

Is the number of words an indicator for ambiguity?

We were curious about the effect of length of a requirement statement on ambiguity. The longer the requirements the more prone it is to syntactical ambiguity. However the length could also indicate that extra information was presented that would help to interpret the requirement. For this analysis the requirements were partitioned by requirement size in 8 groups ranging from 6 to 45 words. Each group has a span of 5 words. Figure 5 shows that the more words were used the more interpretations were given by the review panel. The average word count for unambiguous requirements was 11, the average word count for ambiguous requirements with two interpretations was 20. It is tempting to say that requirements should be written as short as possible, but it could very well be that the requirements were so lengthy because the message being expressed was complicated. What we can learn is that lengthy requirements demand extra attention.

Will disambiguation decrease the different interpretations among stakeholders?

Five randomly selected requirements were rewritten with the help of the rules described in Berry *et al.*[4]. The selected requirements were found to be ambiguous by both the review panel and by the systematic review. Furthermore, the average word count of these requirements was 25. Rewriting these five requirements took about half a day, however this did not include a check that the new description expressed the intention of the customer. The rewritten requirements were reviewed again by the same protocol as specified in subsection 3.2.2.

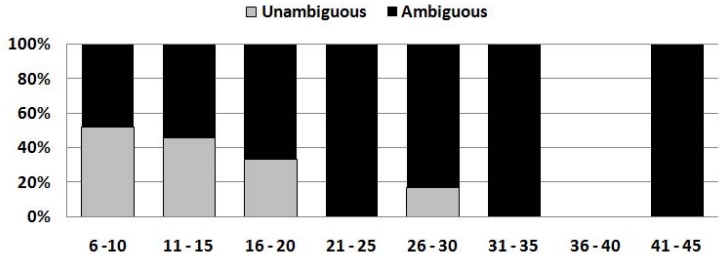


Fig. 5. x-axis: requirements with no. of words. y-axis: % ambiguous vs unambiguous.

This test found that four of the five requirements were unambiguous. The fifth requirement was still ambiguous, caused by a vague word. Disambiguating required mandating a specific solution, limiting the design space more than the customer required.

The rewritten requirements contained more words than the original requirements. In fact, the average word count was 46 for the rewritten requirements, the requirements contained more and shorter sentences. The review panel members mentioned they had no problems in comprehending the rewritten requirements. This shows that length is not the most important indicator for ambiguities.

8 Conclusion

In this research we studied the effect of a highly ambiguous requirements document on project success. The studied project was the development of a complex system that took about 21 man years to develop and was canceled after an independent test team found over 100 blocking issues. The perception of the contractor was that many of these issues were caused by ambiguity in the requirements. Independent tests by humans showed that 91% of the requirements were ambiguous. An automated test revealed that 92% of the requirements were ambiguous. A root cause analysis on 40 of the main issues showed that only one of the examined issues was caused by ambiguous requirements. This issue was resolved and explained 2% of the budget overrun.

In this project, ambiguous requirements were not the main cause of the issues found by the external test team, and cannot explain the failure of the project. Both the independent test team and the third party development team found ways to cope with the high level of ambiguity.

We can only speculate to the reason why the project was canceled. As a fixed price project it wasn't because of the budget overrun. Studying the bug reports we saw that the number of open defects and newly found defects remained high throughout the acceptance test. This resulted in a loss of confidence in the product by the customer. A possible explanation for most of the issues is that due to schedule pressure not enough care was given to implementation details. Also, as Brooks already knew, adding people to a project that is already late is usually not effective.

9 Future Work

We speculate that a requirements document that is perceived to be of low quality triggers a process of better understanding requirements. It would be interesting to see what the effect is of ambiguity in a requirements document that is perceived to be of high quality. This might result in false confidence that all requirements are correct. The development team will be less inclined to question these requirements and the problems have a bigger chance to go unnoticed. It would be interesting to repeat this research for such projects. It is also interesting to repeat this research in projects where communication is challenged by organizational, structural and political factors. In these cases we speculate that ambiguous requirements will have a bigger impact.

References

1. Abran, A., Moore, J.W., Bourque, P., Dupuis, R.: SWEBOK: Guide to the software engineering Body of Knowledge. IEEE Computer Society, Los Alamitos (2004)
2. Alexander, I.F., Stevens, R.: Writing better requirements. Addison-Wesley, Reading (2002)
3. Berry, D.M.: Ambiguity in Natural Language Requirements Documents. In: Paech, B., Martell, C. (eds.) Monterey Workshop 2007. LNCS, vol. 5320, pp. 1–7. Springer, Heidelberg (2008)
4. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity. Univ. of Waterloo Technical Report (2003)
5. Bouma, G., Van Noord, G., Malouf, R.: Alpino: Wide-coverage computational analysis of Dutch. In: Computational Linguistics in the Netherlands 2000. Selected Papers from the 11th CLIN Meeting (2001)
6. Campbell, M.J., TDV Swinscow: Statistics at square one. John Wiley & Sons, Chichester (2002)
7. Chantree, F., Nuseibeh, B., de Roeck, A., Willis, A.: Identifying nocuous ambiguities in natural language requirements. In: 14th IEEE International Conference Requirements Engineering, pp. 59–68 (2006)
8. Davis, A., et al.: Identifying and measuring quality in a software requirementsspecification. In: Proceedings of First International Software Metrics Symposium, pp. 141–152 (1993)
9. Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., Ruggieri, S.: Achieving quality in natural language requirements. In: Proceedings of the 11 th International Software Quality Week (1998)
10. Fabbrini, F., Fusani, M., Gnesi, S., Lami, G.: An automatic quality evaluation for natural language requirements. In: Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ, vol. 1, pp. 4–5 (2001)
11. Gause, D.C.: User DRIVEN design - The luxury that has become a necessity. In: A Workshop in Full Life-Cycle Requirements Management. ICRE (2000)
12. Hull, E., Jackson, K., Dick, J.: Requirements engineering. Springer, Heidelberg (2005)
13. Kamsties, E.: Surfacing ambiguity in natural language requirements. PhD thesis, Fachbereich Informatik, Universitat Kaiserslautern, Kaiserslautern, Germany (2001)
14. Kamsties, E., Berry, D.M., Paech, B.: Detecting ambiguities in requirements documents using inspections. In: Proceedings of the first Workshop on Inspection in Software Engineering (WISE 2001), pp. 68–80 (2001)
15. Lauesen, S.: Software requirements: styles and techniques. Addison-Wesley, Reading (2002)
16. Mullery, G.: The perfect requirement myth. Requirements Engineering 1(2), 132–134 (1996)
17. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: ICSE 2000: Proceedings of the Conference on The Future of Software Engineering, pp. 35–46. ACM, New York (2000)
18. Robertson, S., Robertson, J.: Mastering the requirements process. Addison-Wesley Professional, Reading (2006)
19. Schneider, G.M., Martin, J., Tsai, W.T.: An experimental study of fault detection in user requirements documents. ACM Transactions on Software Engineering and Methodology (TOSEM) 1(2), 188–204 (1992)
20. Schramm, W.: How communication works, p. 51. Mass Media & Society (1997)

On the Role of Ambiguity in RE

Vincenzo Gervasi[1,2] and Didar Zowghi[2]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Faculty of Eng. and Inf. Technology, University of Technology, Sydney, Australia

Abstract. [**Context and motivation**] *Ambiguity* has long been pictured as one of the worst enemies of the specifier, especially with reference to ambiguity in natural language (NL) requirements specifications. [**Question/problem**] In this paper, we investigate the nature of ambiguity, and [**Principal ideas/result**] advocate that the simplistic view of ambiguity as merely a "defect" that has to be avoided at all costs does not do justice to the complexity of this phenomenon. We also provide a finer classification of several types of ambiguities, distinguishing their different causes and effects in the development process. [**Contribution**] This better understanding can help in the analysis of practical experiences and in the design of more effective methods to detect, mark and handle ambiguity.

Keywords: Ambiguity, natural language, abstraction, absence.

1 Introduction

The study of properties of software requirements specifications (SRS) has been an important and recurring theme throughout the evolution of requirements engineering (RE) research. Fundamental issues concerning the *contents* of requirements, such as how to avoid or detect inconsistencies in SRS (and how to remove or tolerate them), or how to ensure completeness of the requirements, have been a mainstay in RE. The reasons are clear: no implementation can satisfy an inconsistent SRS, and an incomplete SRS, once implemented, will not satisfy all the needs of the users.

Another related stream of research has been concerned with properties of the *form* of requirements. Properties such as understandability, conciseness, etc. have been studied and discussed, and techniques to ensure an SRS exhibits such properties (or to identify and fix their negative dual properties) have been proposed.

In this paper, we focus mainly on *ambiguity*, i.e. the phenomenon by which multiple distinct meanings can be assigned to the same requirement (or, more generally, sets of requirements), and discuss the relationships between ambiguity and certain other phenomena which are often observed

We do not provide in this work advice on how to avoid introducing ambiguity in an SRS, nor on how to remedy it when it is detected. Rather, we focus on understanding what ambiguity *is* (with particular reference to its role in RE), on

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 248–254, 2010.

how, when and by whom it is introduced in SRS, and on what the effects of its various forms are. Armed with this understanding, we discover that ambiguity is not necessarily a defect, and in fact can play an important positive role both in the requirements as a document, and in the requirements elicitation process.

2 Ambiguity and Interpretation

Ambiguity is a complex, multi-level phenomenon. While the general concept of "having multiple meanings" is relatively easy to describe, locating the original source – or root cause – of the ambiguity may be more challenging. Moreover, ambiguity may or may not be detected by the several parties involved in requirements elicitation or analysis, and could be intentional or accidental; its extent can be confined to a minor detail or encompass some major aspect of the system.

It is clear that the simple intuitive definition of "having multiple meanings" is insufficient for a deep understanding of ambiguity. We will instead use as reference frame that of the classical denotational approach, where semantics is given by a *function* mapping from a *source domain* (the text of the requirements) to a *target domain* (the denotation of their semantics), which can be arbitrary (e.g., input/output semantics, performances, development cost estimates, etc.). In requirements engineering, all these three elements are fuzzy at best. The source domain can include, in addition to written text, spoken information, observed behavior, etc. The target domain should in theory be such that it is possible to determine if a given implementation satisfies the requirements, but in practice it is often vague in itself. And finally, the mapping is ill-defined, and often – even when a strict formal definition exists – may well be misunderstood by at least some of the stakeholders (e.g., an end-user will probably be incapable of understanding the meaning of a fragment of Z from a complex requirements specification). This last point is worth stressing: for the purpose of assessing the effects of ambiguity, it is not the *intrinsic* meaning of a requirement or set thereof

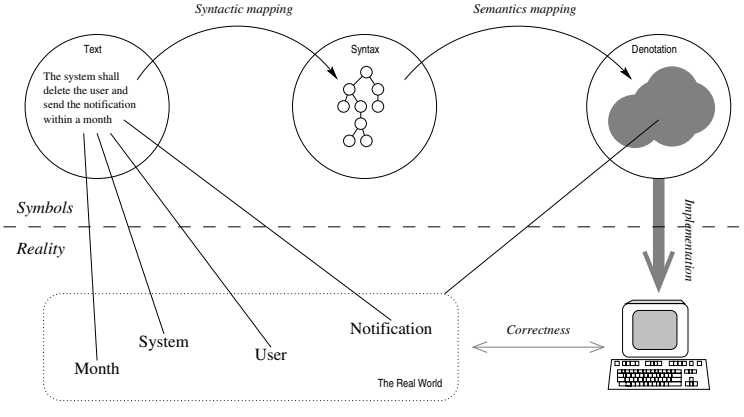


Fig. 1. The theoretical framework for the occurrence of ambiguity

that is of interest (even when we have such a thing, e.g. in formal languages), but the *interpretation* placed on it by a cognitive agent or interpreter.

Figure 1 shows the various transformations which can lead to multiple meanings. The denotation of the semantics of the requirements is then what drives the implementation, whose purpose is to build a computer-based system which will interact with its environment in such a way that the original intent is satisfied.

3 Sources of Ambiguity

Ambiguity is essentially a linguistic phenomenon, thus it is appropriate to analyze its sources according to the usual paradigm of lexicon, syntax, semantics (we omit pragmatics due to space constraints). We briefly outline the main issues here, not delving into all the details.

Lexical level. Ambiguity in lexicon occurs typically when the same term is used to denote different things. This can be an inherent feature of the language being used (for example: homonyms in NL), or happen even in more formal languages due to lack of or imprecise *designations*. In fact, even in formal languages such designations are invariably rooted in the informal "real" world, and all stakeholders must *a-priori* agree on their meaning (thus establishing a common base of reference). In Figure 1, terms appearing in the requirement, such as "user" or "month" are just lexical tokens. They can correspond to different designations, e.g. "month" could mean a 30-days period, or a 31-days period, or till the same-numbered day in the next month, etc.[1] Without a more precise designation, the term "month" is seriously ambiguous: for example, which date is "a month from today"?

Syntactic level. Ambiguity on the syntactic level is easier to define. It stems from there existing multiple parse trees for a sentence; to each possible parse tree, a different meaning is attached, hence the ambiguity. In Figure 1, multiple possible parse trees exist for our sample requirement. In fact, the sentence could be parsed as "The system shall delete the user and (send the notification within a month)" or as "The system shall (delete the user and send the notification) within a month", where the parentheses have been used to indicate the two critically different parsings.

Semantic level. Semantic ambiguity happens when the source text is uniquely determined in both lexicon and syntax, and still multiple meanings can be assigned to the sentence. In this case, the ambiguity lies not in the source, but in the function assigning meaning to the source, labeled in Figure 1 as the *semantics mapping* function. In Figure 1, even if we have precise designations for "month", "system", "user" etc., and even if we are told which of the two syntactic interpretation to take, we could still have doubts on the intended semantics. For example, "shall send a notification" means the system will attempt to do it, but how? Is it sufficient to print out a form and hope that someone will put it in an envelope and mail it? What if the notification is sent, but not delivered? Is there some sort of

[1] The Bahá'í calendar, for example, has 19 months of 19 days each, plus 4 intercalary days (5 in leap years) which are not part of any month.

acknowledgment to be expected? Maybe the notification could be sent via a text message to the user's mobile phone? And so on (endlessly).

4 Ambiguity, Abstraction, Absence

We have seen in the previous section how even a simple sentence like our example

The system shall delete the user and send the notification within a month.

which could appear among the requirements, say, for a library loan system when membership expires, is actually riddled by various types of ambiguity, so that its correct implementation, missing further information, is probably beyond hope.

One could then believe that ambiguity is thus a pernicious defect, to be eradicated with ruthless determination from any self-respecting requirements specification. Unfortunately, this noble determination often leads to the practical impossibility of writing, analyzing, and implementing, the requirements for even the simplest of software systems, while huge amounts of effort are devoted to writing beautifully complex and extensive specifications. We believe instead that ambiguity can also play a positive role in requirements specifications, beyond its well-known "political" role in negotiations (which we consider to be a form of pragmatic ambiguity). To this end, we need first to distinguish among three related concepts:

Ambiguity is the existence of multiple semantics denotations for the same source text. Whether this is caused by syntactic ambiguity (Fig. 2, right) or by semantics ambiguity (Fig. 2, left), or by lexical or pragmatic issues (or any combination thereof) is irrelevant: the essence of the phenomenon is in having multiple (distinct) semantics for the same source. Ambiguity has often been considered a defect in requirements, in account for the lack of a single, well-defined, shared semantics that can be used to drive implementation and verification.

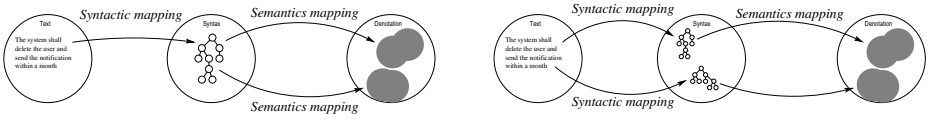


Fig. 2. Cases of ambiguity: semantics ambiguity (left), syntactic ambiguity (right)

Abstraction is the omission of some details (or more properly, of some information content). Ambiguity can be used as a means of abstraction, in that the omitted detail is the information needed to discriminate between multiple semantics in order to identify the "right" one (in the eye of the requirement author). Abstraction is generally considered a desirable quality in requirements, up to a point, in that it avoids overspecification and simplifies the requirements, keeping them manageable and allowing stakeholders to focus on the important parts.

Absence is the total lack of information on some specific aspect; as such, it is the extreme case of abstraction, where certain information content is abstracted to nothingness. Being a special case of abstraction, absence as well can be related to ambiguity as discussed above. Absence is the major motivation for requirements elicitation: knowledge "holes" are usually considered dangerous in specifications, and need to be filled-in by investigating the problem and its domain in more depth.

Our sample requirement also contains instances of abstraction and absence. Using the term "month" can be seen as a not very precise way to refer to some specific duration of time, essentially conveying the idea of "I don't care about the *exact* duration, but it should be close to 30 days", that being a case of abstraction. At the same time, nothing is said about the actual contents of the "notification", e.g. which text should be sent. Of course, the implemented system will have to send some specific text (we cannot keep the message abstract), so the missing information is needed, and hence this is really a case of absence.

Since ambiguity can play both negative and positive roles, the question arises naturally: when is "bad" ambiguity turned into "good" abstraction, and when is the latter turned again into "bad" absence? We believe this question, in this crude form, is too simplistic, and more about the *intentions* of the stakeholders working on and with the requirements must be considered.

As a first step, let us identify two roles in working with requirements, those of *author* and of *reader*. The author is the stakeholder that commits a requirement to a written form; the reader is a participant to the development process who needs the information conveyed by the requirements in order to perform his or her own job. Both writers and readers may or may not recognize the ambiguity which is present in a requirement. This gives rise to the combinations shown in Table 1.

Table 1. Recognized and unrecognized ambiguity

		Reader	
		recognized	unrecognized
Writer	recognized	**(a)** ambiguity used by writer as abstraction device, recognized as such: good use of ambiguity, any implementation correct.	**(b)** writer used ambiguity as abstraction device, reader only recognized one possible meaning: loss of design space.
	unrecognized	**(c)** writer wrote ambiguous requirement without realizing, reader assumed all meanings are acceptable: potential incorrect implementation.	**(d)** ambiguity gone unnoticed: if both reader and writer agree on meaning, correct implementation "by chance", otherwise incorrect implementation.

We assume here for simplicity that recognizing an ambiguity means being cognizant of all the possible meanings, whereas not recognizing it means considering only one meaning (which may or may not be the intended one), and not realizing that there is a potential ambiguity. Naturally, in practice we can be

faced with fuzzy cases, in which we suspect there is an ambiguity but cannot determine for certain.

When the ambiguity is recognized by the writer (cases (a) and (b) in Table 1), we can assume that it is *intentional*: the writer uses ambiguity as a means of abstracting away unnecessary details, signifying that all possible meanings are all equally acceptable to her as correct implementations of the requirements. In our example the clause "within a month" could be intentionally ambiguous, meaning that the writer (e.g., the customer) is not interested in the exact limit, as long as there is a fixed term, and the term is *approximately* a month. In case (a), the reader (e.g., the implementor) also recognizes the ambiguity, and is free to choose, among all possible implementations that satisfy the requirement in any of its possible ambiguous meanings, the one that best suits him: for example, a simple `limit=today()+30;` in code will suffice. In case (b), the reader may not realize that the writer has given him freedom to implement a vague notion of *month*, and might implement a full calendar, taking into account leap years and different month lengths, possibly synchronizing with time servers on the Internet to give precise-to-the-second months, etc. The resulting implementation will be correct, but unnecessarily complex. The design space for the solution has been restricted without reason, and maybe opportunities for improving the quality of the implementation in other areas (e.g., robustness or maintainability) have been lost.

If the ambiguity is not recognized by the writer (cases (c) and (d) in Table 1), we can assume it is *not intentional*: in a sense, it has crept in against the writer's intention. Hence, only one of the possible meanings is correct, whereas others are incorrect. The implementation can still be correct, but only by chance (because, among the possible interpretations, the correct one was chosen). Moreover, when multiple readers are involved, as is the case in every real-life project, the chances of *every* reader taking up the same correct interpretation is slim: so, this type of ambiguity will probably lead to a wrong implementation, or to a correct implementation which is tested against the wrong set of test cases, or to a correct implementation which is tested correctly but then erroneously documented in users' manuals according to a wrong interpretation, etc.

The critical issue becomes: how can one be certain if a given instance of ambiguity is intentional or not? The answer lies in a generalized concept of *markedness*. In linguistics, a normal, default form (the one which more *naturally* would be used) is considered unmarked, whereas a non-standard form is considered marked (the more un-natural the form is, the more marked it is considered). For example, instead of a more natural (and unmarked) "within a month", a requirement could be written as "within a period of approximately one month". This second form is less naturally occurring, hence more marked, and thus provides evidence that the ambiguity was intended by the writer (one could also say that it dispels ambiguity by explicitly stating vagueness).

NL does not offer a specific way of marking intentional ambiguity from unintentional one, but conventions could be established to that effect. Notice that using more contrived forms (e.g., adding "approximately") does constitute a case

of markedness, but it is totally unsystematic, and thus cannot be relied upon by the reader. In contrast, what would be needed is a systematic marking of ambiguity.

5 Conclusions

We presented an analysis of the role of ambiguity in requirements, discussing how ambiguity is intimately linked to two other relevant phenomena (abstraction and absence of information), and how it can play a positive role in requirements authoring and analysis when used in conjunction with the concept of markedness to convey abstraction. In fact, we argue that each instance of ambiguity is not useful or damaging, nocuous or innocuous, "good" or "bad" just by itself, but that these characteristics can only be defined with reference to a particular set of stakeholders.

We believe that an improved understanding of the nature and effect of ambiguity can help clear the way for a more positive view of ambiguity in requirements, and suggest ways to improve the current state of practice. In particular, tools and techniques aimed at identifying instances of ambiguity in requirements could incorporate the classification presented, and focus more on assessing naturalness and markedness in the NL form of requirements, in addition to identifying inherently vague terms or ambiguous parsing structures. The final goal would be assisting their users focus on identifying and properly handling the different types of ambiguity, particularly the really critical and risky cases.

Towards a Framework to Elicit and Manage Security and Privacy Requirements from Laws and Regulations

Shareeful Islam[1], Haralambos Mouratidis[2], and Stefan Wagner[3]

[1,3] Institut für Informatik, Technische Universität München, Germany
{islam,wagnerst}@in.tum.de
[2] School of Computing, IT and Engineering, University of East London, Great Britain
haris@uel.ac.uk

Abstract. [**Context and motivation**] The increasing demand of software systems to process and manage sensitive information has led to the need that software systems should comply with relevant laws and regulations, which enforce the privacy and other aspects of the stored information. [**Question/problem**] However, the task is challenging because concepts and terminology used for requirements engineering are mostly different to those used in the legal domain and there is a lack of appropriate modelling languages and techniques to support such activities. [**Principal ideas/results**] The legislation need to be analysed and align with the system requirements. [**Contribution**] This paper motivates the need to introduce a framework to assist the elicitation and management of security and privacy requirements from relevant legislation and it briefly presents the foundations of such a framework along with an example.

Keywords: Security requirements, privacy requirements, Secure Tropos, modelling, and evolving legislation.

1 Introduction

Software systems are now widely used for applications including financial services, industrial management, and medical information management. Therefore, it is now necessary that software for critical applications must comply with the relevant legislation. Sensitive system information must not be open to unauthorised access, processing, and disclosure by legitimate users and/or external attackers. This situation makes security to one of the key components involved in ensuring privacy [1]. Information security and data privacy laws are in general complex and ambiguous by nature and in particular relatively new and evolving [2, 10].

Such laws often undergo evolution to support the demands of the volatile world. Several factors such as the introduction of new restrictions, regulation mandates to increase security, privacy and quality of service, technology evolution, and new threats and harms are commonly responsible for the amendment of legislation. An amended legislation enforces an organization to review their internal policies and to adopt the changes in their software systems. Especially legally relevant requirements (security and privacy in our case) should be adapted to avoid corresponding risks. Therefore, research should be devoted to the development of techniques that

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 255–261, 2010.

systematically extract and manage requirements from laws and regulations in order to support requirements compliance to such laws and regulations. We believe evolution at requirements level is critical in order to meet the needs of its stakeholders and the constraints such as legal requirements so that change can be traced further through the life cycle. Due to the above situation, the elicitation of legally compliant requirements is a challenging task.

This paper, as an extension of our previous work [9], discusses the need to introduce a framework to allow the elicitation and management of security and privacy requirements from relevant laws and regulations and it briefly presents the foundations of a novel framework that assists in eliciting security and privacy requirements from relevant legislation and it supports the adoption of changes in the system's requirements to support the evolution of the laws and regulations. Our contribution addresses the current research problem of handling evolution of laws, regulation and their alignment to the requirements.

2 Overview of the Framework

The framework is based on the Secure Tropos modelling language [4, 5] and goal-driven security risk management (GSRM) [8]. It includes four main activities and each consists of several steps that support the purpose of the activity and produces artefacts. One of the main input elements required for performing the activities are relevant legal texts. Therefore business specifications including business goals, process, and an initial set of user requirements are required to identify the relevant legal text. Figure 1 shows an overview of the framework with the input documents, activities, and steps in the activity, artefacts produced from the activities, and the associated links.

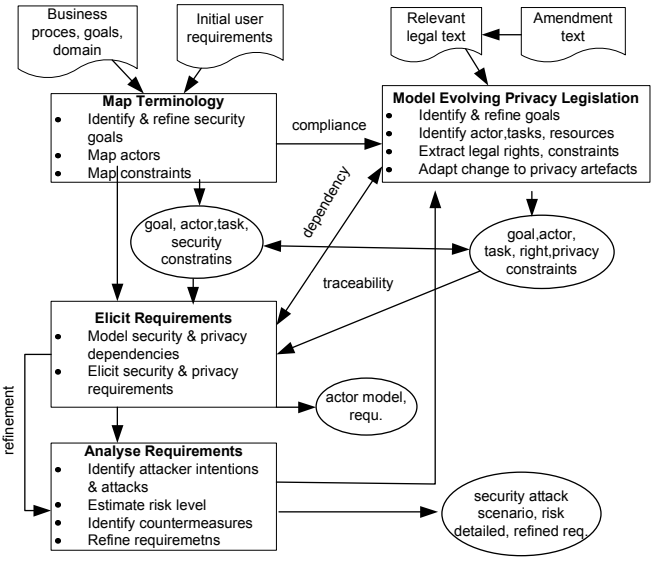


Fig. 1. Overview of the framework

Activity 1: Model Evolving Regulation. The first step in that activity is to identify and refine the goals from the privacy legislation by analysing why the regulation and specific sections of the regulation were introduced to support the specific context. We follow a basic legal taxonomy proposed by Hohfeld [11] to identify the terms of privacy legislation. The taxonomy is based on legal rights and classifies in several elementary concepts including privilege, claim, power, immunity, duty, no-right, liability, and disability. The next step involves the identification of the relevant actors, their performed tasks, and the required resources in the system environment to support the goals. Legal rights are concerned with the actions that the actors are allowed or permitted to perform [10, 11]. The rights should focus on certain *consent*, *enforcement*, *notice*, *awareness*, and *participation* relating to the privacy taxonomy [1]. We use activity and purpose patterns [10] along with a sub-set of the Secure Tropos language to support these steps [4]. The final step involves the adoption of privacy artefacts with the legislation evolution. We consider the privacy artefacts identified previously to support the analysis of the requirements' change and we structure our analysis into three possible ways, with which legal text evolves [7]: addition of a new clause, modification of an existing clause, deletion of a clause.

Activity 2: Map Terminology. During this activity legal terms are mapped to the terms used for security and privacy requirements. In particular, the legal artefacts identified from the previous activity are systematically mapped to the security artefacts. An initial step is to identify and refine the security goals. Security goals are identified by analysing the business and initial user requirements of the system environment, and by following the privacy taxonomy [1]. The main focus is to ensure critical security properties such as confidentiality, integrity, availability, authenticity, and non-repudiation as well as the privacy goals from the previous activity within the overall system environment. Once the goals are identified, the next step is to map the actors from the legal concepts to the security concepts by following both security and privacy goals. Finally we need to map the privacy and security constraints for the goal satisfaction by following goals, actors, and task.

Activity 3: Elicit Requirements. During the first step of this activity, we model the secure and privacy dependencies through the Secure Tropos actor model [4], by following the identified actors, goals, tasks, and constraints. This allows us to establish the compliance link from the legal concepts to security concepts. Finally security and legal requirements are identified by elaborating both security and privacy constraints and traceability from legal concepts to security is attained through the identified artefact; in particular by following the relevant goals, tasks, and actors.

Activity 4: Analyse Requirements. This final activity refines the initial requirements by following risk and evolution techniques. Security threats and privacy harms that obstruct the relevant goals and influence the relevant non-compliance issues are identified and analysed. To support the analysis, we combine goal-driven risk management [8] with Security Attack Scenarios (SAS) [5]. The activity starts by identifying the attacker's intentions and attacks. This allows us to identify the potential resources of the system that might be attacked. In our framework, we model the goals of an attacker, attacks and possible resources of the system that might be attacked with an extended set of *attack links* [5]. The next step of the activity is to estimate the risk level based on the analysis techniques of GSRM so that risks are categorised as *high*,

medium, and *low* by focusing on the risk likelihood and impact. Once the risks are estimated then it is important to identify the countermeasures to prevent the potential attacks and non-compliances issues. Finally the initial requirements are refined (if needed) to accommodate provisions for the countermeasure of attacks that cannot be prevented with the existing set of requirements.

3 Example

The presented example briefly illustrates the applicability of our framework to a specific application context, where a German bank that offers its customers use of a smart card (EC card) for payments. We have chosen relevant privacy regulations by considering the EU directive 95/46/EC [6] and German Federal Data Protection Act (FDPA) [3] that are related for the context. In the text below, normative phrases (such as "must", "shall") and conditional phrases (such as "and", "or") are in bold; a subject for an action is underlined; an action is italicized; an object is in bold and underlined; a measurement parameter is in bold, italicized, and underlined.

Directive 95/46/EC, Article 17 (partial), Security of processing (partial)

1. Member States shall provide that the controller must implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a network, and against all other unlawful forms of processing. Having regard to the state of the art and the cost of their implementation, such measures shall ensure a level of security appropriate to the risks represented by the processing and the nature of the data to be protected.

German Federal Data Protection Act, Annex (partial)

1. To prevent unauthorised persons from gaining *access* to data processing systems with which ***personal data*** are *processed* **or** *used* (access control).

Activity 1: Model Evolving Regulation. The goal of 95/46/EC is to ensure *personal data protection*, which is refined with *security in processing* and supported by *appropriate technical and organisational measures* in article 17. The FDPA supports the goal of 95/46/EC by including high level requirements such as *access control* in its annex. The *customer* and *application providers* are the two main actors. *Customer data* is the main resource, which contains personally identifiable information such as the customer name and sensitive information such as card and account details. The resource is shared for common tasks such as *collect customer data*, and *update account balance*. Among the identified legal rights is that the providers have the liability to take appropriate measure to ensure privacy protection and to protect from any accidental and unlawful activities. To simplify the illustration of our framework, at this stage, we have not considered any evolution of legal texts but we consider it during the analysis activity below.

Activity 2: Map Terminology. The security goal for the application context is already considered by the legal goals. Therefore, we directly refine the goals to support the security properties. For example, *access control* is refined to *identification and adequate authorisation*. Goals such as *data integrity* and *secure communication* as well as tasks like *providing customised reports about balance* are necessary for this

context. To map actors, for simplicity, we consider high level actors such as bank and card issuer and assume their roles support the security constraints. The security constraints supported by the actors are: *only legitimate customer*, *keep communication secure*, *transfer minimum data*, and *preserve anonymity*. Finally, security and privacy constraints are mapped to align with the goals, such as providers' liability to consider any technical measure as privacy constraints and only legitimate customer, keep communication secure as security constraints support goals like access control, and secure communication.

Activity 3: Elicit Requirements. Once the security and privacy constraints are analysed, this activity initially models their dependencies and then elicit relevant requirements such as; i) The customer shall be identified and authenticated before allowed to perform any transaction through the card; ii) The bank shall only provide the minimum of required data to the retailer that supports the business purpose.

Activity 4: Analyse Requirements. Finally, the elicited requirements are analysed based on the security threat, privacy harm, and legislation evolution. We consider *data retainment* from directive 2006/24/EC [6] as evolution by adding new constraints from the legislation to the application context.

Article 6 partial (Periods of retention)
Member States shall ensure that the **categories of data** specified in Article 5 are *retained* for periods of not less than six months and not more than two years from the date of the communication.

The amendment of the legal text introduces the bank's liability to retain the customer data for a certain period to time. At this stage, we need to identify the attacker intentions and attacks for the non-compliance issues in the environment. Among the several attackers' goals, we consider here *obtain sensitive data,* by external attackers through unauthorised access to the system or eavesdropping, and by internal attackers through misuse. Furthermore, amendment of legislation also supports the attacker's goal, as the longer data is retained, the higher the likelihood of accidental disclosure, data theft, and other illegal activities. Commonly the impacts of the factors are high once the attacker successfully performs any attack. Therefore, for simplicity we consider the risk level as high for both high and medium likelihoods of the risk factors. Finally, requirements are refined such as, the data shall be categorised in a manner that some sensitive data would not transfer even to the trusted business partners, and new requirements are elicited, such as "The system shall preserve the customer categorised data for the minimum amount of time to support the business purpose and to meet the legal compliance" to ensure security and privacy goals.

4 Related Work

Mouratidis et al. [4] presented Secure Tropos for eliciting security requirements in terms of security constraints and the approach of Islam [8] extended it with security attack scenarios, where possible attackers, their attacks, and system resources are modelled. Islam [8] also proposed a goal-based software development risk management model (GSRM) to assess and manage risks from the RE phase. Antón et al. [1]

introduce two classes of privacy related software requirements through two classes: privacy protection goals such as integrity & security and privacy harms based on vulnerabilities relating to information monitoring, aggregation, storage, transfer, collection, and personalization. Breaux et al. [10] consider activity, purpose, and rule sets to extract rights, obligations, and constraints from legal texts. Ghanavati et al. [7] use User Requirement Notation based on Goal-oriented Requirement Language for a requirement management framework by modelling hospital business process and privacy legislation in terms of goals, tasks, actors, and responsibilities. Siena et al. [2] focus on Hohfeld's legal taxonomy and map the legal rights with the i* goal modelling language to extract legal compliance requirement. In [8], we use Secure Tropos to model regulation, based on Hohfeld's legal taxonomy, in order to extract requirements that comply with legislation.

As foundation for our work we use SecureTropos, GSRM, activity and purpose patterns, and rule sets. Our framework contributes that it enables the analysis of privacy regulations beyond the only permitted and required actions and it facilitates the consideration of non-compliance issues and risk management since the early stages of the development process. Furthermore, it supports adopting security and privacy requirements to a change of legislation.

5 Conclusion

Security and privacy practices are important for software that manages sensitive information and for stakeholders when selecting software or service providers to serve their business needs. Therefore, organisations responsible to manage sensitive data cannot escape the obligation to implement the requirements established by privacy regulations and changes therein. This paper advances the current state of the art by contributing the foundations of a framework that aligns security and privacy requirements with relevant legislation.

References

[1] Antón, A., Earp, J., Reese, A.: Analyzing website privacy requirements using privacy goal taxonomy. In: Proc. of the IEEE Joint International Conference on RE, pp. 23–31 (2002)

[2] Siena, J., Mylopoulos, A., Susi, A.: Towards a framework for law-compliant software requirements. In: Proc. of the 31st International Conference on Software Engineering (ICSE 2009), Vancouver, Canada (2009)

[3] Bundesdatenschutzgesetz - Federal Data Protection Act (as of November 15, 2006)

[4] Mouratidis, H., Giorgini, P.: Secure Tropos: A Security-Oriented Extension of the Tropos Methodology. International Journal of Software Engineering and Knowledge Engineering. © World Scientific Publishing Company

[5] Mouratidis, H., Giorgini, P.: Security Attack Testing (SAT) - testing the security of information systems at design time. Inf. Syst. 32(8), 1166–1183 (2007)

[6] Information society, Summary of legislation, European Commission

[7] Ghanavati, S., Amyot, D., Peyton, L.: A Requirements Management Framework for Privacy Compliance. In: Workshop on Requirements Engineering (WER 2007), Toronto, Canada (2007)
[8] Islam, S.: Software development risk management model: a goal driven approach. In: Proceedings of the Doctoral Symposium for ESEC/FSE on Doctoral Symposium, Amsterdam, The Netherlands (2009)
[9] Islam, S., Mouratidis, H., Jürjens, J.: A Framework to Support Alignment of Secure Software Engineering with Legal Regulations. Journal of Software and Systems Modeling (SoSyM) Theme Section NFPinDSML (to appear 2010), doi:10.1007/s10270-010-0154-z
[10] Breaux, T.D., Antón, A.I.: Analyzing Regulator Rules for privacy and Security Requirements. IEEE Transactions on Software Engineering 34(1) (January-February 2008)
[11] Hohfeld, W.N.: Fundamental Legal Conceptions as Applied in Judicial Reasoning. Yale Law of Journal 23(1) (1913)

Visualizing Cyber Attacks with Misuse Case Maps

Peter Karpati[1], Guttorm Sindre[1], and Andreas L. Opdahl[2]

[1] Department of Computer and Information Science
Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
{kpeter,guttors}@idi.ntnu.no
[2] Department of Information Science and Media Studies,
University of Bergen, Bergen, Norway
Andreas.Opdahl@uib.no

Abstract. [**Context and motivation**] In the development of secure software, work on requirements and on architecture need to be closely intertwined, because possible threats and the chosen architecture depend on each other mutually. [**Question/problem**] Nevertheless, most security requirement techniques do not take architecture into account. The transition from security requirements to secure architectures is left to security experts and software developers, excluding domain experts and other groups of stakeholders from discussions of threats, vulnerabilities and mitigations in an architectural context. [**Principal idea/results**] The paper introduces *misuse case maps*, a new modelling technique that is the anti-behavioural complement to use case maps. The purpose of the new technique is to visualize how cyber attacks are performed in an architectural context. [**Contribution**] The paper investigates what a misuse case map notation might look like. A preliminary evaluation suggests that misuse case maps may indeed make it easier for less experienced stakeholders to gain an understanding of multi-stage intrusion scenarios.

Keywords: security, requirements elicitation, misuse case, use case map, misuse case map.

1 Introduction

Much effort in the security area focuses on *surveillance* and *fire-fighting*, which are undoubtedly crucial aspects of the "cops and robbers" game of security. A complementary approach is *prevention by design*. Instead of detecting and mitigating attacks, prevention by design strives to eliminate security vulnerabilities in the early phases of software development. Vulnerabilities can take many shapes. One time, a well-known, long-used mechanism is misused in an unexpected way. Another time, an obscure part of the software system is exposed and exploited by an attacker. In order to eliminate vulnerabilities early during software development, it is essential to understand the attackers' perspectives and their ways of working, as pointed out by many authors (e.g., [1-3]).

Much research has been performed on modelling the technical aspect of complex attacks, targeting security experts and security-focused software developers. Our premise is that secure software development may benefit from involving a wider

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 262–275, 2010.

group of stakeholders, such as domain experts (who know the subject and usage worlds of the proposed software system) and regular software developers who have no special security training. Ideally, this would happen during the requirements phase of software development, which is a common ground for domain experts, software experts and security experts to meet. Also, clarifying security issues already during the requirements phase results in a security conscious design that saves many troubles (money, effort, time, reputation etc.) later on.

There are already several techniques and methods available for dealing with security requirements in the early software development phases. But there is no technique or method that addresses security requirements in relation to design of secure architectures. In practice, however, the two cannot be completely separated: possible threats will depend on the chosen architecture; the choice of architecture might depend on what threats are considered the most dangerous ones; different architecture choices offer different mitigations strategies etc. Our idea is that domain experts, regular software developers and others should be allowed and encouraged to reason about security concerns in an architectural context. For this purpose, suitable representations are needed that combine user, designer and security perspectives on the proposed software system, so that all stakeholders can understand the issues and contribute their ideas and background knowledge to the security discussions.

Hence, the purpose of this paper is to introduce a new attack modelling technique that combines an attacker's behavioural view of the proposed software system with an architectural view. The technique is intended to be useful for a variety of stakeholders. The technique is called *misuse case maps (MUCM)*. It is inspired by *use case maps* [4, 5], into which it introduces anti-behaviours. The technique is illustrated through a multi-stage intrusion from the literature [6]. Results from a preliminary evaluation are also reported.

The rest of the paper is organized as follows. We present related work in Sec. 2. Misuse case maps are introduced in Sec. 3. Misuse case maps are applied to an example from the literature in Sec. 4. A preliminary evaluation is presented in Sec. 5. Finally, we conclude and point out future directions for our work in Sec. 6.

2 Modelling Techniques for Security Requirements

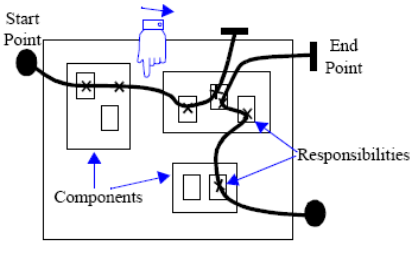
2.1 Security Requirements

There are already many techniques and methods available that focus on elicitation and analysis of security requirements during early RE. Attack trees [7] and threat trees [8] are trees with a high level attack (or threat) at the root, which is then decomposed through AND / OR branches. Secure i* [9] is an extension of the i* modelling language, where malicious actors and their goals are modelled with inverted variants of the usual icons. Abuse frames [10], extend problem frames with anti-requirements that might be held by potential attackers. Abuse cases [11], misuse cases [12], and security use cases [13] are security-oriented variants of regular use cases. Abuse and misuse cases represent behaviours that potential attackers want to perform using the software system, whereas security use cases represent countermeasures intended to avoid or repel these attacks. The difference between abuse and misuse cases is that the

latter show use and misuse in the same picture, whereas abuse cases are drawn in separate diagrams. We will return to misuse cases in Sec. 2.4.

There are also techniques and methods that attempt to cover later development phases. Secure Tropos [14] extends the Tropos method with security-related concepts, whereas KAOS has been extended with anti-goals [15]. The CORAS project [16] combined misuse cases with UML-based techniques into a comprehensive method for secure software development. Other security-focused extensions of UML include UMLsec [17] and SecureUML [18]. Languages for secure business process modelling have also been proposed based on BPMN [19] and UML activity diagrams [20]. Security patterns describe recommended designs for security [21], and the formal specification language Z has been used to specify security-critical systems [22, 23].

Despite the many techniques and methods available for dealing with security in the early phases of software development, there is so far no technique or method that links security requirements and architecture. There is, however, a technique that links software functionality in general with architecture, which we now present.



Imagine tracing a path through a system of objects to explain a causal sequence, leaving behind a visual signature. Use Case Maps capture such sequences. They are composed of:

- **start points** (filled circles representing pre-conditions or triggering causes)
- causal chains of **responsibilities** (crosses, representing actions, tasks, or functions to be performed)
- and **end points** (bars representing post-conditions or resulting effects).

The responsibilities can be bound to **components**, which are the entities or objects composing the system.

Fig. 1. Notation and interpretation of UCMs (from [4])

2.2 Use Case Maps (UCM)

The *use case map (UCM)* notation [4,5,24,25] was introduced by Buhr and his team at Carleton University in 1992. It quickly gained popularity. UCMs have been used in both research and industry, in particular in the telecommunications sector. It is a part of the User Requirements Notation (URN) standardized by the International Telecommunication Union (ITU).

UCMs provide a combined overview of a software system's architecture and its behaviour by drawing usage *scenarios paths* (aka use cases) as lines across boxes that represent architectural run-time *components*. The boxes can be nested to indicate hierarchies of components. The scenario paths are connected to the components they run across by *responsibilities* drawn as crosses. Fig. 1 illustrates and explains the basic UCM notation. This UCM shows multiple scenarios as multiple paths across the architecture components.

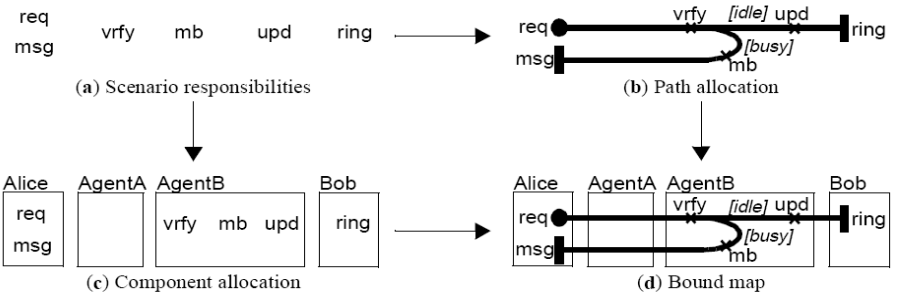


Fig. 2. Variants of UCMs (from [4])

Fig. 2 shows how a UCM binds responsibilities, paths, and components together. In this simple example, "…a user (**Alice**) attempts to call another user (**Bob**) through some network of agents. Each user has an agent responsible for managing subscribed telephony features such as Originating Call Screening (**OCS**). Alice first sends a connection request (**req**) to the network through her agent. This request causes the called agent to verify (**vrfy**) whether the called party is idle or busy (conditions are between square brackets). If he is, then there will be some status update (**upd**) and a ring signal will be activated on Bob's side (**ring**). Otherwise, a message stating that Bob is not available will be prepared (**mb**) and sent back to Alice (**msg**)." [4] The example also shows how sections of scenario paths can be split and joined to indicate alternative or parallel paths.

In this manner, UCMs offer high-level views for software and systems development [4, 24]. They combine an architectural overview with behavioural detail and thus facilitate discovery of problems within collections of scenarios or use cases. UCMs can also serve as synchronization means among the scenarios/use cases to check them for completeness, correctness, consistency, ambiguity or consistent abstraction levels. UCMs provide several additional notations for visualizing more complex behaviours and more refined relationships between scenarios and architecture components. We do not present all of them here. The UCM notation also offers variants that use only two of the three core components (scenario paths, architecture components and responsibilities) [4, 25]. In particular, paths and components can be used without responsibilities for presenting very-high level overviews and for "napkin-type" sketching of ideas.

2.3 Use Case Maps and Anti-functional Requirements

Security has been discussed in relation to UCMs in connection with *performance-related completions* ("additions, including annotations, component insertions, environment infrastructure, deployment, communication patterns, design refinements and scenario or design transformations which correspond to a given deployment style") [26] and in connection with *RE for data sharing in health care* in [27]. UCMs are used to represent an example of *early aspects* at the requirements level in [28]. Security appears there as a MUCM component, first in the main UCM and later in a plug-in (or sub-map) of the main UCM. However the aim of the example is to demonstrate the UCM notation and not to address security as such.

Beyond these contributions, we have not found any direct considerations of the *security* perspective in UCMs. But there are two contributions that address *safety* in a UCM context. We review them here because security and safety requirements are both examples of *anti-functional requirements*, i.e., requirements that state what the software system should *not* do. Hence they are similar to functional requirements in that they are both concerned with the software system's behaviour, but they have opposite *modalities*. Wu and Kelly [29] present an approach to derive safety requirements using UCMs. The approach aims to provide assurance on the integrity of requirements elicitation and formulation. First they formulate the problem context in their process, followed by analysis of deviations, assessment of risks, choice of mitigations and formulation of safety requirements. The initial set of requirements is refined iteratively while a software system architecture is also developed incrementally. The authors conclude (1) that UCMs are effective for capturing the existing architectural context (structure and specific operational modes) beside the intended behaviour and (2) that the explicit architectural references extend the scope of the deviation analysis compared with the one over functions or use cases. In [30], Wu and Kelly extend their approach into a negative scenario framework (along with a mitigation action model), which has a wider theoretical background and is more general than the proposal in [29]. The UCM no longer plays the central role, and the approach to identifying deviations is less specific. Although their framework targeted safety-critical systems, the authors suggest that it is applicable for other systems as well, such as security- and performance-critical ones.

Despite the interest in combining UCMs with anti-functional requirements, no representation technique has so far been proposed that provide a combined overview of the attackers' and the architects' views of a proposed software system. However, there is a technique that shows how to extend and combine representations of wanted software system behaviour, as covered by UCMs, with an attacker's attempts to cause harm, which we now present.

2.4 Misuse Cases (MUC)

Misuse cases (MUC) [12] extend use cases (UC) for security purposes with *misusers*, *misuse cases* and *mitigation use cases*, as well as the new relations *threatens* and *mitigates*. They represent security issues by expressing the point of view of an attacker [31]. Whereas regular UCs describe functional requirements to the software system, MUCs thereby represent *anti-functional requirements*, i.e., behaviours the software should prohibit. They thus encourage focus on security early during software development by facilitating discussion between different stakeholder groups, such as domain experts, software developers and security experts. MUCs have also been investigated for safety [32-34] and other system dependability threats [35] and compared with other similar techniques like FMEA, Attack Trees and Common Criteria [33, 36, 37]. MUCs can be represented in two ways, either diagrammatically or textually. Diagrammatically, MUC symbols inverts the graphical notations used in regular UC symbols, and UC and MUC symbols can be combined in the same diagram. Textually, both lightweight and an extensive template are offered [12, 34].

3 Misuse Case Maps (MUCM)

3.1 The Need for Misuse Case Maps

The previous section shows that there are many techniques and methods available for dealing with security requirements in the early software development phases, both during RE and in the transition to later phases. But there is no technique or method that addresses security requirements in relation to design of secure architectures. Yet it is well known that requirements and architecture can rarely be considered in complete isolation. Contrarily, architecture is essential for security in several ways. The types of architecture components suggest typical weaknesses and attack types for the component (e.g., a router can be scanned for open ports). The specifics of architecture components suggest specific weaknesses (e.g., a particular router model is likely to have a particular standard password). The path each function takes through the software architecture suggests which general and specific weaknesses a user of that function might try to exploit. Furthermore, when weaknesses have been identified, architectural considerations are equally important for mitigating the threats. To alleviate these and other problems, there is a need for a security requirements technique that combines an attack-oriented view of the proposed software with an architectural view.

3.2 Basic Concepts

MUCMs extend regular UCMs for security purposes with *exploit paths* in much the same way that MUCs extend regular UCs with *misuse cases*. As in regular UCMs, the *exploit paths* in a MUCM are drawn across nested boxes that represent hierarchically-organized architecture *components*. In addition to regular *responsibilities*, the intersection of an exploit path and a component can constitute a *vulnerability*, which is a behavioural or structural weakness in a system. A component can be a *vulnerability* too. A threat combines one or more weaknesses to achieve a malicious intent. Vulnerabilities can be *mitigated*, where a mitigation counters a threat and translates to a security requirement. Both regular scenario paths and exploit paths can be combined in the same MUCM, just like a MUC diagram can also show UCs.

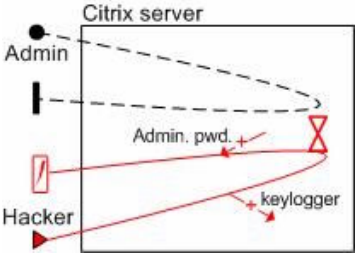


Fig. 3. A simple MUCM example

Fig. 3 shows an excerpt of a MUCM comprising one component, a server, along with a regular scenario path and an exploit path. The excerpt is part of the bigger example presented in Sec. 4. We will reveal further details of the notation below.

3.3 Notation

The MUCM notation is based on the regular UCM notation, just like the MUC notation is based on the UC notation. The MUC notation uses inversion of use-case symbols to distinguish wanted from unwanted behaviour. This is not easy to do for use case maps, where the start and end points of regular scenario paths are already shown with filled icons and where the paths themselves are drawn as whole lines. Instead, we have explored using colours and different icon shapes to distinguish between wanted and unwanted behaviours in MUCMs.

The leftmost column of Fig. 4 shows the basic MUCM symbols. An exploit path starts with a *triangle* and ends in a *bar* (as in UCMs) if no damage happened. Otherwise the path ends in a *lightning* symbol. Exploit paths can be *numbered* to show the order of stages in a complex intrusion, where the stages will mostly be causally related, in the sense that each of them builds on the results of previous ones. A *vulnerable* responsibility (e.g., an authentication point) or component (e.g., an unpatched server) is indicated by a closed curve, and a *mitigation* of the resulting threat (e.g., secure password generation, routines for patching servers) is shown by shading the interior of the closed curve. *Responsibilities* can be left out whenever they are not relevant from the intrusion's point of view. Through these basic symbols, MUCMs offer a basic notation that is close to the simplified UCM notation suggested for very-high level overviews and for "napkin-type" sketching of ideas. We expect this to be the most prominent use of MUCMs in practice.

Yet at this early stage it is worth exploring more detailed notation alternatives too. For example, the rightmost column of Fig. 4 shows how a time glass can be used to indicate that an exploit path must *wait* for a regular scenario path to reach a certain point. The example in Fig. 3 used this notation to show how an attacker, who have secured access to a Citrix server at an earlier intrusion stage, installs a keylogger on the server in order to snatch the administrator's password. The hour glass indicates that the attacker has to wait for an administrator to log in before the keylogger can snatch the password.

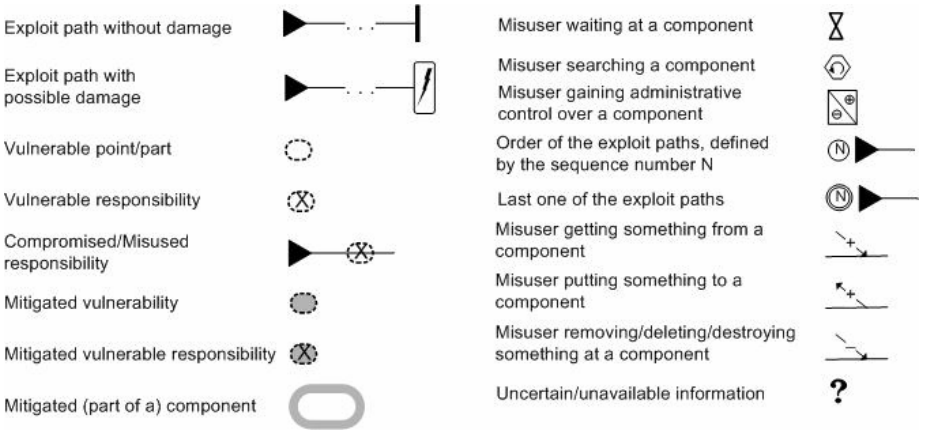


Fig. 4. The proposed MUCM notation, with the basic symbols shown on the left and further tentative extensions on the right

Get, *put* and *remove* arrows can be used to show how an exploit path interacts with a component. An example involving the *get* arrow is when the attacker accesses the password hash files. An example of a *put* arrow is when the attacker installs a sniffer program on one of the servers. An example using the *remove* arrow is when the attacker deletes his/her traces from a system. We will see in the complex example that not all the information is available about the case. Similarly, when (re)creating an intrusion, some parts of it may be unclear at first. The question marks can be used as reminders about unclear issues.

Labels can be attached to symbols. For example, a label at the start of a UCM path might indicate the *role* of the actor if it affects a connected exploit path; a label at the end of an exploit path might be labelled with the *result* of the exploit if it is not clear from the path alone; and *get*, *put* or *remove* arrows can be labelled with the types of data or software that are accessed. These arrows are part of the regular UCM notation as well, where they have a slightly different meaning. Hence, their interpretation depends on the context. Further work should consider using distinct symbols, such as a wave arrow, to differentiate the notations.

Like in regular UCMs, the granularity of the intrusion representation can change by combining or exploding steps. Consider a case where the attacker downloads a file of password hashes from one machine, cracks them in his/her own computer and proceeds to log into another machine with a cracked password. This could be shown as individual steps or as a composite step – a MUCM *stub* – that hide the cracking process and leads the exploit path from the first machine with the hashes to the one logged into with the cracked password.

As already explained, we expect the basic MUCM symbols to be the ones most used in practice, and we expect the notation we suggest here to evolve further. In this paper, however, we will stay with the symbols we have used in the preliminary evaluation to be presented in Sec. 5.

4 A Complex Intrusion Example

MUCMs and the associated notation have been developed through a series of modelling exercises using hacker stories from [6]. We have so far developed MUCMs for 5 of the 12 relevant intrusion cases described there, 3 of them in full detail. We present one of them here to illustrate MUCMs and the first version of the associated notation.

4.1 The Bank Intrusion Case

The bank intrusion is a multi-stage intrusion presented in [6, Chapter 7]. We suggest following the intrusion on the MUCM in Sec. 4.2 while reading through the intrusion steps.

First, the intruder found an interesting bank by browsing a web site with organizations and IP ranges assigned. Next, he probed for further details about the IP addresses of the bank and found a server that was running Citrix MetaFrame (remote access software). He then scanned other networked computers for the remote access port to Citrix terminal services (port 1494). The attacker knew he might be able to enter the server with no password, as the default Citrix setting is “no password

required". He searched every file on the computer for the word "password" to find the clear text password for the bank's firewall. The attacker then tried to connect to routers and found one with default password. He added a firewall rule allowing incoming connections to port 1723 (VPN).

After successfully authenticating to the VPN service, the attacker's computer was assigned an IP address on the internal network, which was flat, with all systems on a single segment. He discovered a confidential report written by security consultants containing a list of network vulnerabilities. He also found operation manuals to the bank's IBM AS/400 server on a Windows domain server. The default password

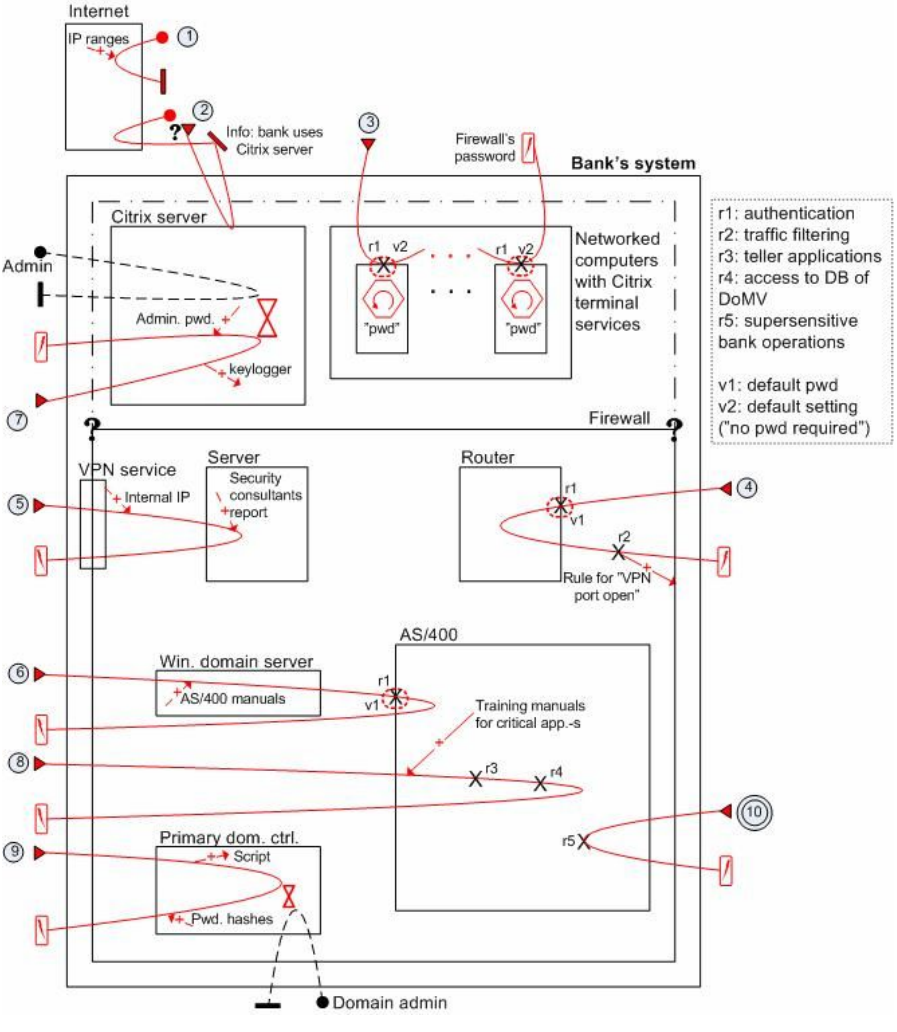


Fig. 5. A misuse case map for the bank intrusion. The whole red line depicts the attacker's footprint whereas the dashed black line shows the regular users' activities.

worked for the AS/400. The intruder installed a keylogger on the Citrix server, waited for an administrator to log in and snarfed the administrator's password. He now had access to training manuals for critical AS/400 applications, giving him the ability to perform any activity a teller could. He also found that the database of the Department of Motor Vehicles was accessible from the bank's site. He accessed the Primary Domain Controller (which authenticates login requests to the domain) and added a disguised script that extracted password hashes from a protected part of the system registry in the administrator's startup folder. He then waited for a domain administrator to log in so the script would be triggered and password hashes written to a hidden file. He then cracked the appropriate password. The most sensitive parts of the bank's operations could now be accessed (generating wire transfers, moving funds etc.). A manual he had already found described how to complete a wire transfer form.

The attacker was a "white-hat" hacker who claimed not to harm the bank or its customers as a result of the intrusion.

4.2 MUCM

Fig. 5 shows a MUCM for the bank intrusion. Some details were omitted, either because they were not given in the original text (e.g., how the access to some of the components was secured) or because the details were intuitive and would only overload the map (e.g., to access the internal computers, the attacker always went through the VPN).

5 Preliminary Evaluation

To preliminary evaluate MUCMs and the MUCM notation, we sent out a written evaluation sheet to more than 20 colleagues and other contacts. We received 12 responses. All respondents had MSc or PhD degrees, except one MSc student who, on the other hand, had professional experience as a system administrator. All the degrees were in computing. 6 of the respondents were working as academics, 4 in industry and 2 in both academia and industry.

The evaluation sheet had three sections. The first section explained the aim and the required conditions of the experiment. There was no time limit, but the respondents were asked to perform the evaluation without interruption. The second section gave an introduction to UCM and MUCM. The third section included a copy of the textual description of the bank intrusion from [6], along with the corresponding MUCM (Fig. 5). The third section also comprised three sets of questions, regarding (a) the *background* of the participants, (b) the participants' *understanding of the case*, and (c) the *user acceptance* of the technique. The user acceptance questions were inspired by the Technology Acceptance Model (TAM) [38], reflecting the three TAM-variables *perceived usefulness*, *perceived ease of use* and *intention to use* with 2 items each, giving six items (or questions) in all. At the end of the sheet, open comments were invited. In addition, the respondents were asked how much time they spent on the evaluation sheet and which aids they relied on when answering the questions about understanding of the case (either the *textual description*, the *misuse case map* or *memory*).

The participants spent between 20 and 60 minutes on the task (37 minutes on average). We split the responses in the following four groups, depending on which aids they reported to have relied on when answering the questions about *understanding of the case*. The four groups were TD (9 valid responses relying on the *textual description*), MEM (6 valid responses relying on *memory*), MUCM (6 valid responses relying on the *misuse case map*) and NON-M (4 valid responses *not* using the *misuse case map*). Because most respondents reported relying on more than one aid, the groups overlap considerably. Nevertheless, a comparison of the responses gives a useful first indication of the strengths and weaknesses of MUCM as an aid for understanding a complex intrusion scenario.

Table 1 summarizes the responses according to group. The TD and MUCM groups spent most time on the task, the MEM and NON-M groups the least. With regard to background experience, the groups were quite similar, with average scores between 3.6 and 3.9 on a scale from 1 to 5 (with 5 being highest). The TD and MUCM groups reported slightly lower experience on average than the MEM and NON-M groups, suggesting that less experienced respondents relied more on external aids. This may explain in part why they used more time too. The MUCM group had the highest average percentage (77%) of correct answers to the questions about understanding, whereas the NON-M group had the lowest one (68%). Although time may have played a part, we take this to be an indication that MUCM may indeed be a beneficial aid for understanding complex intrusions. The TD group also had a high average score on understanding, and the MEM group a lower one.

Table 1. Responses to the evaluation sheet, grouped according to the aids used when answering the questions about understanding of the case

Group	Responses	Time (min)	Back-ground	Under-standing	PU	PEOU	ITU
TD	9	36	3.6	74%	3.8	2.9	3.9
MEM	6	30	3.9	71%	3.9	2.5	3.8
MUCM	6	37	3.7	77%	3.9	3.3	3.9
NON-M	4	30	3.8	68%	3.9	2.5	4

On average, the four groups rated the perceived usefulness (PU) of MUCMs similarly, from 3.8 to 3.9 (again on a scale from 1 to 5 with 5 being highest). The average ratings on perceived ease of use (PEOU), however, were considerably higher for MUCM (3.3) than for NON-M (2.5), suggesting that perceived ease of use played an important role in the respondents' decisions whether to use the MUCM or not when answering questions about the case. The TD group also had a high average rating and MEM a low one, on PEOU. In general, the scores on PU were higher than for PEOU, indicating that the somewhat elaborate MUCM notation used in the evaluation was perceived as complex. On intention to use (ITU), however, the average ratings were nearly identical between the groups (from 3.8 to 4.0). Surprisingly, intention to use

misuse case maps in the future was highest for the NON-M group that had not used MUCMs at all. Because the evaluation was preliminary, we do not address validity issues here.

We received written and oral comments on the following issues: the notation was hard to understand although it could become easier to read with time; the map contained too much detail; the map contained too little detail; UML sequence diagrams may be a better alternative; the component concept is unclear because it mixes physical and logical entities; and MUCMs are good for analysis but maybe not for communication. We plan to address these comments in the further development of the technique.

6 Conclusions and Future Work

The paper has introduced a new attack modeling technique, *misuse case maps (MUCM)*, that combines an attacker's behavioral view of the proposed software system with an architectural view. The purpose of misuse case maps is to offer a representation technique with the potential to include a wider group of stakeholders, such as domain experts and regular software developers, in security considerations already during the earliest development phases. The technique and its notation was illustrated through a multi-stage bank intrusion described in the literature. Results from a preliminary evaluation were also reported, indicating that MUCM may indeed be a beneficial aid for understanding complex intrusions.

Of course, the preliminary evaluation is severely limited. It used only a small example, which precluded statistical analysis. The evaluation was not controlled, and the subjects were colleagues and other contacts who might have been positively biased towards our proposal. Hence, further empirical evaluations are clearly needed, for example investigating different complex intrusion scenarios for the future like in [39, Chapter 3]. They should involve more subjects working under more controlled conditions.

Future work on MUCMs should address issues such as how to avoid overly complex, spaghetti-like maps, how to best communicate intrusions to domain experts and regular software developers, and how to involve them in the vulnerability exploring and mitigating process. Further evaluations and practical studies will use MUCMs in increasingly realistic settings. We intend to combine MUCM with other attack modelling and security analysis techniques. We also plan to provide practical guidelines to establish a security requirements method and provide tool support for it. The method should perhaps be further extended to consider *anti-functional requirements* in general, addressing *safety requirements* in particular in addition to security. Important questions to address will be when and how to apply the security and safety experts' knowledge and how to manage the different types of information that is generated from the cooperation between customers, domain experts and developers.

Acknowledgement. This work was performed as part of the ReqSec project (www.idi.ntnu.no/~guttors/reqsec/) and funded by the Norwegian Research Council.

References

1. Barnum, S., Sethi, A.: Attack Patterns as a Knowledge Resource for Building Secure Software. In: OMG Software Assurance Workshop (2007)
2. Koziol, J., et al.: The shellcoder's handbook: discovering and exploiting security holes. John Wiley & Sons, Chichester (2004)
3. Hoglund, G., McGraw, G.: Exploiting Software: How to Break Code. Addison-Wesley, Boston (2004)
4. Amyot, D.: Use Case Maps Quick Tutorial (1999), http://www.usecasemaps.org/pub/UCMtutorial/UCMtutorial.pdf
5. Buhr, R., Casselman, R.: Use case maps for object-oriented systems. Prentice-Hall, Inc., Upper Saddle River (1995)
6. Mitnick, K.D., Simon, W.L.: The art of intrusion: the real stories behind the exploits of hackers, intruders & deceivers. Wiley, Chichester (2005)
7. Schneier, B.: Secrets & lies: digital security in a networked world. John Wiley & Sons, Chichester (2000)
8. Amoroso, E.G.: Fundamentals of computer security technology. Prentice-Hall, Inc., Upper Saddle River (1994)
9. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: Proc. RE 2003, vol. 3, pp. 151–161 (2003)
10. Lin, L., et al.: Using abuse frames to bound the scope of security problems (2004)
11. McDermott, J., Fox, C.: Using abuse case models for security requirements analysis (1999)
12. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Engineering 10(1), 34–44 (2005)
13. Firesmith, D.J.: Security use cases. Technology 2(3) (2003)
14. Giorgini, P., et al.: Modeling security requirements through ownership, permission and delegation. In: Proc. of RE, vol. 5, pp. 167–176 (2005)
15. Van Lamsweerde, A., et al.: From system goals to intruder anti-goals: attack generation and resolution for security requirements engineering. In: Requirements Engineering for High Assurance Systems (RHAS 2003), vol. 2003, p. 49 (2003)
16. Dimitrakos, T., et al.: Integrating model-based security risk management into eBusiness systems development: The CORAS approach. In: Monteiro, J.L., Swatman, P.M.C., Tavares, L.V. (eds.) Proc. 2nd Conference on E-Commerce, E-Business, E-Government (I3E 2002), pp. 159–175. Kluwer, Lisbon (2002)
17. Jurjens, J.: UMLsec: Extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002)
18. Lodderstedt, T., et al.: SecureUML: A UML-based modeling language for model-driven security. In: Jézéquel, J.-M., Hussmann, H., Cook, S., et al. (eds.) UML 2002. LNCS, vol. 2460, pp. 426–441. Springer, Heidelberg (2002)
19. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Towards an integration of security requirements into business process modeling. In: Proc. of WOSIS, vol. 5, pp. 287–297 (2005)
20. Rodriguez, A., Fernandez-Medina, E., Piattini, M.: Capturing Security Requirements in Business Processes Through a UML 2.0 Activity Diagrams Profile. In: Roddick, J., Benjamins, V.R., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R.A., Grandi, F., Han, H., Hepp, M., Lytras, M.D., Mišić, V.B., Poels, G., Song, I.-Y., Trujillo, J., Vangenot, C. (eds.) ER Workshops 2006. LNCS, vol. 4231, pp. 32–42. Springer, Heidelberg (2006)

21. Schumacher, M., et al.: Security Patterns: Integrating Security and Systems Engineering. Wiley, Chichester (2005)
22. Boswell, A.: Specification and validation of a security policy model. IEEE Transactions on Software Engineering 21(2), 63–68 (1995)
23. Hall, A., Chapman, R.: Correctness by construction: Developing a commercial secure system. IEEE Software, 18–25 (2002)
24. Buhr, R.J.A.: Use case maps for attributing behaviour to system architecture. In: 4th International Workshop of Parallel and Distributed Real-Time Systems (1996)
25. Buhr, R.J.A.: Use case maps as architectural entities for complex systems. IEEE Transactions on Software Engineering 24(12), 1131–1155 (1998)
26. Woodside, M., Petriu, D., Siddiqui, K.: Performance-related completions for software specifications. In: 24th International Conference on Software Engineering (2002)
27. Liu, X., Peyton, L., Kuziemsky, C.: A Requirement Engineering Framework for Electronic Data Sharing of Health Care Data Between Organizations. In: MCETECH (2009)
28. Mussbacher, G., Amyot, D., Weiss, M.: Visualizing Early Aspects with Use Case Maps. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 105–143. Springer, Heidelberg (2007)
29. Wu, W., Kelly, T.P.: Deriving safety requirements as part of system architecture definition. In: Proceedings of the 24th International System Safety Conference, Albuquerque (2006)
30. Wu, W., Kelly, T.: Managing Architectural Design Decisions for Safety-Critical Software Systems. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 59–77. Springer, Heidelberg (2006)
31. Alexander, I.: Misuse cases: Use cases with hostile intent. IEEE Software 20(1), 58–66 (2003)
32. Sindre, G.: A look at misuse cases for safety concerns. International Federation for Information Processing Publications - IFIP, vol. 244, p. 252 (2007)
33. Stålhane, T., Sindre, G.: A comparison of two approaches to safety analysis based on use cases. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 423–437. Springer, Heidelberg (2007)
34. Stålhane, T., Sindre, G.: Safety Hazard Identification by Misuse Cases: Experimental Comparison of Text and Diagrams. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 721–735. Springer, Heidelberg (2008)
35. Sindre, G., Opdahl, A.L.: Misuse Cases for Identifying System Dependability Threats. Journal of Information Privacy and Security 4(2), 3–22 (2008)
36. Diallo, M.H., et al.: A comparative evaluation of three approaches to specifying security requirements. In: Proc. REFSQ 2006, Luxembourg (2006)
37. Opdahl, A.L., Sindre, G.: Experimental comparison of attack trees and misuse cases for security threat identification. Information and Software Technology 51(5), 916–932 (2009)
38. Davis, F.D.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS quarterly 13(3), 319–340 (1989)
39. Lindqvist, U., Cheung, S., Valdez, R.: Correlated Attack Modeling, CAM (2003)

How Do Software Architects Consider Non-Functional Requirements: A Survey

David Ameller and Xavier Franch

Universitat Politècnica de Catalunya (UPC)
{dameller,franch}@essi.upc.edu

Abstract. [Context and motivation] Non-functional requirements (NFRs) play a fundamental role when software architects need to make informed decisions. Criteria like efficiency or integrity determine up to a great extent the final form that the logical, development and deployment architectural views take. **[Question/problem]** Continuous evidence is needed about the current industrial practices of software architects concerning NFRs: how do they consider them, and what are the most influential types in their daily work. **[Principal ideas/results]** We ran a web survey addressed to software architects about these issues. We got 60 responses that give some light to the questions above. **[Contribution]** Some empirical data has been gathered from industry. The results of this survey may serve as input for researchers in order to decide in which types of NFRs may be necessary to invest more research effort.

Keywords: Non-Functional Requirements, Software Architectures, Web Survey.

1 The Survey

The survey has been developed following an iterative methodology. Each iteration has been revised by IT experts and researchers of the area. For the implementation we chose LimeSurvey, an open source project for developing surveys.

For the dissemination of the survey we used two strategies. On the one hand, personal contact with software architects and on the second hand, advertisement in IT communities hosted in common sites such as LinkedIn and Facebook. We have contacted more than 10 software architects and advertised in the International Association of Software Architects (IASA) group. The survey was running during the year 2009.

The survey had questions about software development. Concretely, we asked about the used architectural styles, the type of developed applications, the technologic platforms used in them, and questions about Non-Functional Requirements (NFRs).

In this work we show the results about NFRs and their relationship to the used architectural style, the type of developed application, and the used technologic platform.

R. Wieringa and A. Persson (Eds.): REFSQ 2010, LNCS 6182, pp. 276–277, 2010.

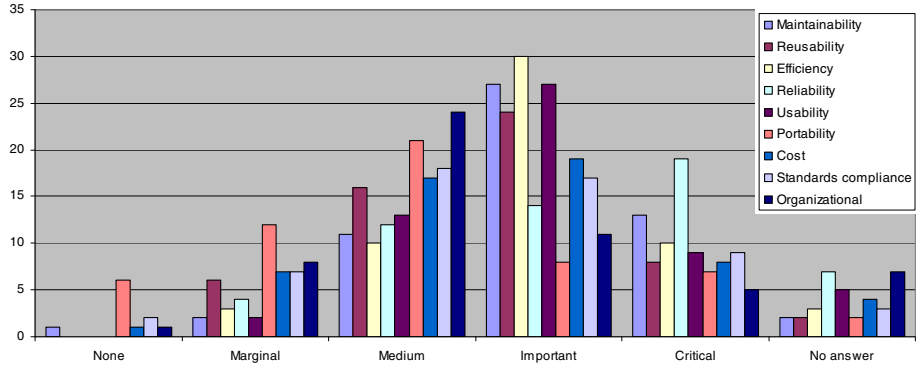


Fig. 1. Importance for architects of the different types of NFRs

2 The Results

We had 60 responses to the survey. The main results of this survey about NFR may be summarized as follows:

- Respondents answered about the importance of NFRs in their habitual software development practices: while 96% of respondents consider NFR (73% at the same level as functional requirements), only 57% use NFR to make architectural and technological decisions.
- Respondents rated nine types of NFRs with respect to the importance to their projects as shown in Fig. 1. Requirements such as maintainability, reusability, efficiency, reliability, and usability have a tendency of being more important for architects than portability, cost, standard compliance, and organizational NFR.
- 80% of respondents declared that the development tools that they use are not well-suited for analysing the compliance with the specified NFRs, whilst 70% would like to have them. For us this is a clear indicator that there is an unsatisfied need in software industry.

Other results (e.g., some relations between NFRs and used architectural styles) were also found when analyzing the data gathered.

3 Conclusions

This survey can be seen as an instrument to show the differences in software development practices between research and industry. In particular, we show the impact of NFRs in the software development practices.

Our position is that a way to obtain empirical evidence about the current state of software architectures usage in IT companies and organizations is asking the involved actors.

Author Index