FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Ein leichtgewichtiger Open-Source-Ansatz für das Anforderungsmanagement

Behebung häufiger Mängel typischerweise verwendeter Werkzeuge und Evaluierung eines Prototypen mit erfahrenen Anwendern

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Michael Jaros
Matrikelnummer 0225848

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: ao. Univ. Prof. Stefan Biffl
Mitwirkung: Dr. Matthias Heindl

Wien, 20/09/2010

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Lightweight Open-Source Tool Support for Requirements Management

Addressing Major Shortcomings of Typically Used ReqM Tools, Development of a Prototype and Evaluation with Experienced Users

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Michael Jaros
Matrikelnummer 0225848

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: ao. Univ. Prof. Stefan Biffl
Mitwirkung: Dr. Matthias Heindl

Wien, 20/09/2010

Erklärung zur Verfassung der Arbeit

Michael Jaros

Laudongasse 58/16

1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20/09/2010

Acknowledgments

I would like to thank Stefan Biffl for his supervision and advice, Matthias Heindl for his constant mentoring and valuable counsel in scientific matters, and Dietmar Winkler for his input on empirical studies.

Furthermore, I want to thank Siemens Austria for the chance to work on an interesting project in an industrial context. Special thanks go to my former superior Franz Reinisch for sharing much of his long-term practical experience in application lifecycle management and giving helpful feedback on the prototype.

I want to thank my friend and colleague Alexander Wagner for the outstanding collaboration at Siemens and in several projects. I thank everyone who took part in the usability test or in the web survey for their important contributions.

Last, but not least, I would like to express my gratitude to my parents Rosa and Herbert and my brother Alexander who have supported me not only in my studies, and I want to direct an especially warm thank you to my partner Eva Maria who is always there for me.

Abstract

Requirements Engineering (RE) deals with elicitation, analysis, documentation and management of requirements. It is a crucial success factor in software development projects. Project Managers and Requirements Engineers need tool support for Requirements Management (ReqM) to enable collaboration on requirements and to keep track of requirements changes, especially in projects that are distributed or have many requirements.

Commercial tools typically have (1) (high) license and training costs, their use is (2) sophisticated and hard to learn; they offer (3) limited integration with other tools and (4) limited extensibility. Most open-source tools are not subject to these limitations and have therefore become increasingly popular in the past few years (e. g. *Subversion*, *Bugzilla*). However, the few existing open-source ReqM solutions suffer from poor state of development and lack of quality. Furthermore, most of the existing ReqM tools have the following shortcomings: (1) Limited versioning without reverting, branching or baselining, (2) inadequate traceability support without user-defined trace types, (3) strongly limited integration with other tools (e. g. configuration and test management, defect tracking).

In order to address these issues, I have developed the open-source ReqM plug-in TreqPro for the collaboration platform Trac. TreqPro provides (1) extended versioning functionality, (2) flexible traceability support, (3) good integration with other tools, and (4) high extensibility.

For evaluating the TreqPro prototype, 8 ReqM experts at Siemens Austria participated in a usability pilot study in which I compared the prototype with the Siemens Austria standard tool Requisite Pro and the open-source tool Trac considering (a) user satisfaction, (b) execution time and (c) completeness for 9 standard use cases. Furthermore, I have created a comprehensive ReqM tool feature catalog as a basis for the prototype development from expert interviews, a web survey and existing work.

In the pilot study, the prototype displayed considerable improvements in all 3 measured parameters (a), (b), and (c) compared to existing tools. Satisfaction could be improved by 40 % compared to Trac and by 61 % compared to Requisite Pro. Execution time could be improved by 34 % compared to Trac and 39 % compared to Requisite Pro. Completeness could be improved by 87 % compared to Trac and 74 % compared to Requisite Pro. Furthermore, the participants gave valuable qualitative feedback on the prototype.

Kurzfassung

Requirements Engineering (RE) beschäftigt sich mit der Ermittlung, Analyse, Dokumentation und Verwaltung von Anforderungen (Requirements) und ist ein wichtiger Erfolgsfaktor für Softwareprojekte. Projektleiter und Anforderungs-Manager benötigen Toolunterstützung für Requirements Management (ReqM), um Zusammenarbeit an Anforderungen zu ermöglichen und Änderungen zu verfolgen, besonders in verteilten Projekten oder solchen mit vielen Anforderungen.

Kommerzielle Werkzeuge haben (1) hohe Lizenz- und Trainingskosten, ihre Benutzung ist (2) kompliziert und schwer zu erlernen, und sie verfügen über (3) unzureichende Integration mit anderen Werkzeugen und sind nur eingeschränkt erweiterbar. Die meisten Open-Source-Werkzeuge unterliegen nicht diesen Einschränkungen und wurden aus diesem Grund in den letzten Jahren immer beliebter (z. B. Subversion, Bugzilla). Allerdings werden die meisten dieser Open-Source ReqM-Lösungen nicht mehr weiterentwickelt bzw. ist die Qualität der Software für den Produktiveinsatz nicht ausreichend. Weiters weisen die meisten ReqM-Tools die folgenden Mängel auf: (1) Eingeschränkte Versionierung ohne Reverting, Branching oder Baselining, (2) unzureichende Unterstützung von Traceability ohne benutzerdefinierte Beziehungstypen, (3) stark eingeschränkte Integration mit anderen Werkzeugen (z. B. Configuration Management-, Test Management-, Defect Tracking-Werkzeuge).

Zur Behebung dieser Mängel entwickle ich die Open-Source ReqM Erweiterung TreqPro für die Plattform Trac, welche (1) erweiterte Versionierungs-Funktionalität, (2) flexible Unterstützung für Traceability, (3) gute Integration mit anderen Werkzeugen sowie (4) hohe Erweiterbarkeit bietet.

Zur Evaluierung des Prototypen nahmen 8 ReqM-Experten bei Siemens Österreich an einer Usability-Pilotstudie teil, in welcher ich TreqPro den bei Siemens Österreich üblicherweise verwendeten Werkzeugen Requisite Pro und Trac gegenüberstellte, wobei (a) Zufriedenheit der Benutzer, (b) Ausführungszeit sowie (c) die Vollständigkeit im Rahmen der Ausführung von 9 Standard-Anwendungsfällen untersucht wurden. Darüber hinaus habe ich in einem umfangreichen Anforderungskatalog für ReqM-Werkzeuge als Grundlage für die Entwicklung des Prototypen Information aus Experteninterviews, einer Web-Umfrage sowie bisherigen Arbeiten zusammengeführt.

In der Pilotstudie wies der Prototyp im Vergleich zu existierenden Werkzeugen signifikante Verbesserungen in allen 3 gemessenen Parametern (a), (b) und (c) auf: Die Zufriedenheit konnte um 40 % im Vergleich zu Trac und um 61 % im Vergleich zu Requisite Pro verbessert warden. Die Ausführungszeit konnte um 34 % im Vergleich zu Trac und um 39 % im Vergleich zu Requisite Pro verbessert werden. Die Vollständigkeit konnte um 87 % im Vergleich zu Trac und um 74 % im Vergleich zu Requisite Pro verbessert werden. Die Teilnehmer gaben für die Weiterentwicklung des Prototypen wertvolles Feedback.

Contents

1 Introduction

This chapter gives an overview on the field of Requirements Engineering (section 1.1), existing ReqM tools and their shortcomings (1.2), the approach of addressing these shortcomings (section 1.3), and the method of evaluation that will be used (section 1.4).

1.1 Requirements Engineering and Requirements Management

This subsection gives an introduction into the nature and importance of requirements (section 1.1.1), and the fields of requirements engineering (section 1.1.2) and requirements management (section 1.1.3).

1.1.1 Requirements

Requirements describe behaviour, constraints and other properties of a software system as well as its context and domain. Requirements usually specify what a system should do instead of how to do it. However it is hard to draw the line and requirements often do contain technical aspects as well. It is possible to specify different types and granularities of requirements, ranging from very general system properties to very specific constraints on a certain operation.

Wrong requirements can lead to the following effects [**Kot98**]:

- The system is delivered late.
- Customers' expectations are not met.
- The system is unreliable.
- Maintenance and improvement costs are high.

1.1.2 Requirements Engineering (RE)

Requirements Engineering (RE) is the process of

> *"[…] discovering, documenting, maintaining a set of requirements for a computer-based system."* [**Kot98**]

An RE process is a structured approach for finding, discussing and documenting requirements. Requirements the stakeholders have agreed upon are written into a requirements specification document and are subject to validation (consistency, completeness).

According to Leffingwell and Widrig [**Lef03**], errors concerning requirements in a software project are both the most common errors and the most expensive errors to fix.

1.1.3 Requirements Management (ReqM)

Requirements Management (ReqM) is

> *… a systematic approach to eliciting, organizing and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system.* [**Lef03**]

According to Kotonya and Sommerville [**Kot98**], change control and change impact assessment are two major ReqM activities.

1.2 Existing ReqM Tools and their Shortcomings

This section describes the existing tools together with their drawbacks. Tools can be divided into FLOSS and proprietary tools. Section 1.2.1, gives an introduction to free/libre open-source software (FLOSS). Section 1.2.2 explains the benefits of using FLOSS in software development. Section 1.2.3 describes proprietary, section 1.2.4 FLOSS ReqM tools. Section 1.2.5 lists the shortcomings of existing ReqM tools.

1.2.1 Free/Libre Open-Source Software (FLOSS)

Free/Libre Open-Source Software is software with a license that allows obtaining, studying, changing and copying its source code. Simplified, the terms free software, libre software and open-source software stand for the same sort of software. People speaking of free or libre software usually prefer pointing out the societal aspects, and people speaking of open-source software mainly mention the economic aspects of FLOSS. The term FLOSS has been created to speak of such software without bias to either of the two groups.

FLOSS is not a new invention. In the 1950s and 1960s it was normal for software users to have the freedoms provided by FLOSS. Towards the end of that period, the software costs started increasing dramatically. Manufacturers began using technical measures to prevent users from studying or copying the source code and copyright law was extended to encompass software. This way, software could be sold separately as a product with a simple business model. In the 1980s, the MIT researcher Richard Stallman started working on the GNU project (which is the basis of several free operating systems) and founded the Free Software Foundation. Together with the Linux kernel developed by Linus Torvalds in 1991, the operating system Gnu/Linux became very popular in the following years. A typical Gnu/linux installation consists of thousands of FLOSS packages maintained by developers from all over the world.

In general, FLOSS did not spread as much on desktop systems as it did on servers: In January 2010, the market share of free web server software is about 70 % [**Net10**]. The market share of free operating systems on desktop computers is only estimated around 1 % [**Wik10b**]. FLOSS is strongly used on mobile and embedded devices. FLOSS operating systems have greater shares on newer markets such as the netbook market (about 30 % in 2009 [**Lai09**]) and the mobile device market (more than 17 % estimated only for Android in 2010 [**Hei10**]).

1.2.2 FLOSS Supporting Software Development

In the last few years, there has been a trend to use FLOSS to support software development wherever possible [**For04**]. Examples for widely-used open-source software:

- Programming Languages: Perl, PHP, Python, Ruby
- Application Frameworks: Spring, Hibernate, Struts, Propel, Cocoon …

- Development Tools: Eclipse, CVS, Subversion, GCC, Make, Ant, Trac …

FLOSS has the following advantages:

- **Ease of Use:** In Genereal, FLOSS is not as overloaded with features as many proprietary products are.
- **Availability:** FLOSS is usually easy to obtain. In many FLOSS distributions, installation and configuration of a FLOSS product can be a matter of just a few commands or mouse clicks.
- **License Costs:** Most FLOSS is not only free as in „free speech“, but also free as in „free beer“, although this is no mandatory implication of a FLOSS license. Even if there are no license costs, there are still training and maintenance costs.
- **Extensibility:** FLOSS can be easily extended. If enough people are interested into a product, they can quickly build up a community where many can contribute to development.

While there are open-source tools for all parts of a software development process, there is almost no open-source software for requirements management (ReqM).

1.2.3 Proprietary ReqM Tools

There are lots of proprietary ReqM tools. These are only a few examples, but there are directories with full listings, see section 2.2.1. A few well-known proprietary tools are IBM Rational Requisite Pro, IBM Rational DOORs, and Borland CaliberRM.

Also, many requirements managers use means like text documents or spreadsheets for smaller projects. These documents are typically created using proprietary standard tools.

1.2.4 FLOSS ReqM Tools

While there are FLOSS tools for all parts of the software development process, there is almost no FLOSS for requirements management (ReqM).

The most notable open-source ReqM tool is the Open-Source Requirements Management Tool (OSRMT) [**Ost10**]. However, development has stalled, and there haven’t been any releases since 2007.

Wikis, Ticket Systems or full collaboration platforms like Trac [**Tra10**] or Redmine [**Red10**] can be used for Requirements Management. Usually, these solutions have major drawbacks, but may work well in small projects.

1.2.5 Shortcomings of Existing ReqM Tools

Existing tools have several shortcomings. In this work, I address 3 of the most important shortcomings which are further described in the following sections.

1.2.5.1 Limited Requirements Traceability

Existing tools provide only limited ways to trace requirements. First of all, there is usually only one single type of link between requirements, but one cannot express different relationships between requirements., e. g. «includes», «is derived from»., «is in conflict with».

Furthermore, it is difficult to maintain traceability if part of the data is managed by a different tool, for example if a requirement should be tested and there are different tools for requirements management and test management.

1.2.5.2 Limited Requirements Versioning

Existing tools only implement basic historization of requirements. This means, that the history of changes is recorded, but the user cannot go back to any old version or branch requirements. Furthermore, only limited baselining support is offered by these tools.

1.2.5.3 Limited Integration with other Tools

Existing tools usually provide complex interfaces to other tools from the same manufacturer, but very little interoperability with other tools. Each tool has its own database and does not share data with other tools.

1.3 Approach for an Improved ReqM Tool

In order to address the shortcomings mentioned in section 1.2.5, I first create a catalog of ReqM tool features as described in section 1.3.1. Using this catalog I build a ReqM tool prototype as described in section 1.3.2. This section gives an overview on the approach which is described in more detail in chapter 3.

1.3.1 Elicitation and Validation of ReqM Tool Features

In order to answer RQ1 and learn about ReqM tool features considered most important, I create an extensive catalog of ReqM tool requirements using existing scientific work, expert interviews, and a web survey among ReqM experts. The creation of the catalog is described in section 3.3.

1.3.2 Development of an OS ReqM Prototype

I develop a ReqM tool prototype as a plug-in for the web-based open-source collaboration platform Trac. The prototype will address the shortcomings of existing tools as explained briefly in the following paragraphs. For more detailed information, please see section 3.4.

1.3.2.1 Addressing Limited Integration

All tools will work as plug-ins to the Trac platform, sharing one single database and one version control repository. The platform will provide basic interconnectivity between its plug-ins and basic services like the formatting of content with wiki syntax.

1.3.2.2 Addressing Limited Traceability

All data involved in the development process will be kept in one single relational database and one version control repository as far as possible. It will be possible to create trace types, so that each artifact can be linked to each other with any trace type.

I will put some effort into the development of an experimental graphical requirements navigation plug-in that will clearly display whole paths of different relations between requirements and other artifacts.

1.3.2.3 Addressing Limited Versioning

Every version of an artifact will be stored together with a link to its predecessor. This way it is not only possible to go back to any version, but also to have two versions with the same predecessor, which can be called a branch.

It will be possible to create baselines which are basically lists of artifact versions. As a baseline is meant to freeze that list of requirement versions, a stored procedure in the database will guarantee the consistency of the baseline.

It will be possible not only to compare arbitrary versions of an artifact, but also to compare whole project versions and easily identify changes.

1.4 Evaluation of the Catalog and the Prototype

I will conduct a survey to complete the ReqM tool feature catalog and to assess its validity.

I will evaluate the TreqPro ReqM prototype in a usability test with ReqM experts. In this usability test I will compare the TreqPro prototype with two existing ReqM solutions regarding user satisfaction, execution time and completeness considering 9 common tasks. In addition, this test will probably generate valuable qualitative feedback on the prototype.

I expect the parts of the concept that are new compared to related work to be a good start for dealing with the shortcomings of existing ReqM software. There will probably be considerable improvement in the three measured parameters (user satisfaction, execution time, and completeness) compared to the Siemens Austria standard tool IBM Rational Requisite Pro.

2 Related Work

This section gives an overview about related work. It has subsections for the field of RQ1 (ReqM tool features) as well as the three main fields of interest for RQ2 (traceability, versioning and tool integration). Introductory material can be found in section 2.1. Work about open-source tools in software development was placed in section 2.6. A list of ReqM tools can be found in section 2.7.

2.1 RE and ReqM Basics

The books listed in this section are standard introductory literature in the field of requirements management.

Dean Leffingwell and Don Widrig give a comprehensive insight into the management of software requirements with use cases [**Lef03**]. Their work is a rather practical guide and explains many techniques from understanding the problem to building and testing the system. It has become a standard reference for practical requirements engineering.

Gerald Kotonya and Ian Sommerville combine a theoretical expedition into the field of Requirements Engineering with an extensive discussion of available techniques [**Kot98**]. Their book is a detailed introductory work on requirements engineering.

Ian Sommerville and Pete Sawyer provide an elaborate guide of best practices [**Som97**] in requirements engineering. The book treats different aspects of requirements engineering. As there are few dependencies between the guidelines, the adaptation to the own company's processes should be easy.

2.2 Sources of ReqM Tool Features

This subsection summarizes related work concerning ReqM tool features in general.

2.2.1 ReqM Tool and/or Feature Databases

Tool feature databases basically offer lists of ReqM tools with an optional search and/or filter functionality. These databases are an important source of available ReqM tool features for creation of a ReqM tool feature catalog (see section 3.3).

The INCOSE tool survey [**INC10**] is a database with web access that allows ReqM tool creators to maintain a list of their tools' features. For visitors, it provides an extensive overview of ReqM tool features and comparison between different ReqM tools.

Ludwig Consulting Services, LLC maintain a website with information about requirements management [**Lud10**]. The site contains a comprehensive list of about 40 ReqM tools, but filtering is not possible, and there are no descriptions.

The NASDAQ stock market maintains a small catalog of important requirements management tools [**NAS10**]. Filters allow finding tools specified by certain criteria.

2.2.2 Literature on ReqM Tool Features

In addition to the tool databases listed above, the research findings on ReqM tool features published by several authors are a source of ReqM tool features for the creation of a ReqM tool feature catalog (see section 3.3). In contrast to the tool feature databases listed in section 2.2.1, these research results allow estimating the importance of each ReqM tool feature. Therefore, the papers listed in this section are of vital importance.

Matthias Hoffmann, Nikolaus Kühn, and Matthias Weber have developed an elaborate catalog of requirements for RM tools [**Hof04**]. Due to their work for the automotive industry, their catalog has not been created especially for software development, but due to their general approach, their work is applicable to other domains but automotive development as well. The following paragraphs summarize core statements of the authors' work as well as the most important ReqM tool features:

The developer should be able to freely define a **Requirements Model**. Different **Views** on the same data should be available. Enrichment of the requirements with **Formatting and External Content** is quite important as well as **Change Management and Comments**,

Documentation of the History, **Baselining** of the current state of requirements, and **Traceability** through user-friendly, semantic, directed linking between requirements. Different methods of analysis of requirements, tool integration, import of requirements specifications. **Document Generation** for official as well as internal purposes should be available. **Collaborative Working on the same data** is as important as checking out data for **Offline Use**. It would be a good idea not to require a client application installed on computers by providing **Web Access** to the RM tool. **Central Installation and Administration** is a central requirement from the project administrators' point of view. Administrators want to have a security concept with **Roles and Permissions**. There should be no **Size Restrictions** on requirement data or user count as limits are hard to pre-estimate. Workflow Management is not considered such an important feature. **Extensibility** is very important to allow reacting to new situations. There are various requirements concerning the **database** (data safety, performance …) and encryption.

Bernd Kretzel has elaborated a list of criteria and requirements for an RE tool as well as a tool selection process at Siemens IT Services and Solutions [**Kre06**]. Although his scope is more general than just on ReqM, the list of required RE tool features can give valuable hints on criteria for a good ReqM tool solution. The author has provided a detailed tree of required features in the following categories:

- Overview of RE Main Activities
- RE as an integrated part of the process
- Requirements Elicitation
- Requirements Analysis and Negotiation
- Requirements Documentation
- Requirements Validation
- Requirements Prioritization
- Requirements Tracing
- Analysis and Reports
- Usability and Administration

Kretzel lists a few essential features of RE tools:

- Centralized requirements storage and management

- History of old requirement versions
- Attributes adding additional information to requirements
- Traces showing relationships between requirements and allowing change impact analysis
- Reports

> *"In general the tools do not support the elicitation step well. Some tools provide e. g. a discussion feature but it seems that the tools are not designed to support this early step of the process. [...] Although the top tools provide nice export features, there are no proper report features. Furthermore, the import feature is often very complex and hard to use."*

On the other hand, the top tools can be easily used for requirement management. RequisitePro excels in simplicity. CaliberRM seems to be like “the big sister” of RequisitePro because it provides similar concepts but more features which make the work with CaliberRM more complex than with RequisitePro. DOORS provides concepts different from those in CaliberRM and RequisitePro. The tool provides even more features as CaliberRM and is therefore also more complex to use.

Kretzl lists several existing tools that proved to work well in various disciplines: CaliberRM, RequisitePro, and DOORS.

Matthias Heindl, Franz Reinisch, Stefan Biffl, and Alex Egyed have evaluated a value based RE tool selection approach developed at Siemens IT Solutions and Services PSE [**Hei06**].

After documenting an RE process description of the RE activities, the authors elaborate a feature tree providing a good overview of RE tool features. The tree (which has also been used in Bernd Kretzl's work [**Kre06**]) is then used as a checklist for evaluation of RE tools, which can be compared with a rating model. A value model helps project managers to rate the importance of features for their projects to find the best tool for their needs.

> *"The approach is straight-forward and seems to be a good means for project managers to compare requirements tools and select the most valuable tool for a particular project."*

As mentioned before, the priorization of ReqM tool features is important for the creation of a ReqM tool feature catalog.

2.3 Importance of Requirements Traceability

In this work, I identify requirements traceability as a feature of ReqM tools in need of improvement. The literature listed in this section substantiates the importance of “process-driven”, continuous traceability for requirements management.

Ramesh et al. point out the importance of traceability for the development of computer systems [**Ram97**]:

> *"A major concern in the development of complex, large-scale computer intensive systems, especially those with evolving requirements, is ensuring that the design of the system meets the current set of requirements. In this context, it is essential to maintain the traceability of requirements to various outputs or artifacts produced during the system design process."*

Finkelstein and Emmerich state that traceability is required to support document-intensive business processes such as requirements management [**Fin00**].

One of the most important early works in the field of requirements traceability was published by Gotel and Finkelstein who have investigated the requirements traceability problem [**Got94**]. They distinguish between pre-RS (requirements specification) traceability and post-RS traceability:

- *"Pre-RS traceability, which is concerned with those aspects of a requirement's life prior to its inclusion in the RS (requirement production).*
- *Post-RS traceability, which is concerned with those aspects of a requirement's life that result from its inclusion in the RS (requirement deployment)."*

According to the authors, there is need for improvement especially in the field of pre-RS traceability. The authors propose increasing modeling of the social infrastructure of requirements production to guarantee the continued ability to “locate and access” requirements contributors and facilitating the contributors’ informal communication.

Ramesh et al. have developed a framework for a traceability scheme and discuss the impact of traceability on quality software engineering [**Ram01**]. They present issues and lessons learned during the introduction of traceability practice in an organization.

The authors point out the need for clearly defined traceability models, which require traceability between requirements and all system components:

> *"In order to achieve this objective, it is essential that traceability be maintained through all phases of the systems* [sic!] *development process, from the requirements as stated or contracted by the customer, through analysis, design, implementation to testing the final product."*

This traceability information can be used

- to prove the fulfillment of requirements,
- to prove that the right design decisions were made,
- for change impact analysis, and
- for maintainance mechanisms.

The authors introduce a framework to describe traceability information. This framework is built on a metamodel and can be used to document traceability practice in a concrete project or organization and to develop similar models in other contexts. The authors tested their approach in a case study.

It turned out that break of traceability in early development stages decreased the effectivity of traceability efforts in later stages. The authors found that providing traceability throughout the development life cycle is difficult if the information interchange between the tools does not work seamlessly.

The authors' results confirm their expectation that traceability can be used to increase software quality.

Hoffmann et al. point out, that a user-friendly implementation of traceability is of high importance [**Hof04**]. Furthermore, they demand the following features of this implementation:

- mandatory links for certain requirements
- clearly visible direction of links from a source to a target
- bidirectional navigability of links
- different link types
- m:n links
- graphical representation of links
- restriction of link vs. requirement types (certain requirement types allow certain link types)
- links across project boundaries

Bala Ramesh and Matthias Jarke [**Ram01**] discuss existing modeling frameworks for organization of traces. They have elicitated the most important kinds of traceability links from their observations and built new models with them. The authors mention four link types, which they describe in detail:

- Satisfaction links
- Evolution links
- Rationale links
- Dependency links

The authors state that the majority of present traceability tools do not offer differentiation between link types.

> *"Moreover, most tools just offer mechanisms for persistent storage and display of traceability information, but they do not support the process of capturing and reusing traces by guidance or enforcement in a systematic manner; those that do tend to have very rigid process models."*

Spanoudakis et al. [**Spa05**] present a roadmap of recent publications related to traceability and issues still open for research. The authors distinguish between manual, semi-automatic and automatic **generation of traceability**. With manual generation, traceability has to be created by the user for each artifact in the application lifecycle which makes it the most ineffective method of establishing traceability due to its high effort for the user and the high probability of errors. Multiple ways of generating traceability in a semi- and full-automatic way have been proposed in order to overcome these limitations. The authors summarize several approaches to semi-automatic generation of traceability: Generation through predefined links, event-based generation and process-driven generation of traceability. **Process-driven generation of traceability** is the approach most interesting for my work because it can establish traceability on the fly during development and fits well into the set of existing tools used for the TreqPro prototype. It currently has a good cost-benefit ratio compared to the manual and the automatic approaches of generating traceability. Current approaches for automatic generation require implementing advanced techniques which are still in an experimental stage and sophisticated to implement.

Gerhart Totz has discussed plugin-based requirements tracing in his master's thesis [**Tot07**]. He has enabled developers to easily create traces from source code to requirements artifacts (across tool borders) from their integrated development environment. Compared to two traditional forms of requirements tracing (Spreadsheet and Requisite Pro), the necessary effort for establishing traceability could be reduced to 1/6. This is interesting for my work because establishing traceability in an easy-to-use way is essential in order to achieve an improvement in this area.

2.4 Importance of Requirements Versioning

In this work, I identify requirements versioning as a feature of ReqM tools in need of improvement. The literature listed in this section substantiates the importance of requirements versioning and baselining for requirements management.

Hoffmann et al. point out, that the "*documentation of the history*" of requirements (to which I refer as **versioning** in this work) is of high importance (++) [**Hof04**]. Furthermore, they demand the following features of this implementation (the importance of each feature is signaled with the symbols "++", "+" and "-" for "high", "medium" and "low" as done by the authors):

- all objects must be versioned (++)

- changes are trackable to smallest data units (++)
- old versions remain available (++)
- reverting to any old version is possible at any time (++)
- change reports (++)
- automatic incrementation of IDs (+)
- major and minor versions are possible (+)
- visualization of changes (+)
- analyze changes to generate information about the project's progress (+)
- change categorization for analysation of changes (+)
- change comments (-)

The authors also mention the importance of **baselining** (++) for freezing the state of the project, e. g. before a major development step or after a review.

The exact checklist of versioning subfeatures a ReqM tool should provide given by the authors is useful for the design of a ReqM tool prototype.

2.5 Importance of ReqM Tool Integration

In this work, I identify the integration of ReqM tools with other tools as a feature in need of improvement. The literature listed in this section substantiates the importance of tool integration for requirements management.

Finkelstein et al. consider the integration of ReqM tools with configuration management and other tools an important field of development for the short-time future [**Fin00**]. The authors also mention distribution and web integration as two medium-term future developments.

Hoffmann et al. describe tool integration as an important ReqM tool requirement and as a prerequisite for complete traceability [**Hof04**]. Furthermore, they demand the following features of its implementation (the importance of each feature is signaled with the symbols "++", "+" and "-" for "high", "medium" and "low" as done by the authors):

- No redundancy must be generated through linking between tools (++)

- The access rights of linked objects must be respected across tool borders (++)
- Linking to smallest possible data structure should be possible (++)
- The tool should allow integration with Configuration Management (++), Test Management (++), Issue Tracking (+), Modeling (+), Communication (+), Performance Analysis (-), Project Management(-)
- Automatic Synchronization between Tools (+)
- Links to external links should be treated the same as internal links (+)
- Full navigability of links (+)
- Support for Integration Platforms (-)
- Connection transparent in both tools (-)

Carey Schwaber (Forrester, Inc.) describes that the focus in application lifecycle management (ALM) should be rather on the connection between tools than on the tools themselves [**For06**]. Schwaber criticizes the use of separate tools and multiple repositories which integrate poorly with each other, which additionally leads to redundancies and inconsistencies. She explains that the efforts spent in maintaining synchronization between tools lower the productivity. Schwaber proposes a shared ALM platform that provides common functionality to the tools (which link into the platform) as well as a unified interface to multiple repositories.

2.6 Open-Source Software Development and Wiki-Based Requirements Management

The works presented in this section underline the importance of using open-source software and processes in software development. They also give requirements and practical guidelines for wiki-based requirements management.

Liz Barnett compares the open-source development model with traditional and agile development models [**For04**]. She presents the open-source approach as a way to encounter the time and budget issues of most software development projects. The author considers it important to adopt some open-source process aspects like team communication, user involvement, automated generation of documentation, building up a knowledge base, and increasing transparency. Involving users in development can facilitate requirements engineering, design and testing. Barnett also proposes collective code ownership which is a large change from the traditionall permission-based approaches – every developer is allowed to change every part of the system.

Barnett's findings align with the general trend to use open-source software in software development projects. The prototype developed in this work will eventually become open-source software. While a typical open-source development process will not be applied instantly (as the prototype is part of a master's thesis), existing open-source components will be used as a basis for the prototype.

In a recent paper, Oezguer Uenalan, Norman Riegel, Sebastian Weber, Joerg Doerr treat the use of wiki-based solutions for requirements management and also specify a few basic requirements for wiki software used for RM [**Uen08**]:

- Classification of requirements
- Specification of a documentation model
- Creation of a versioned and structured specification document out of different artifacts
- Collaboratively review requirements specifications and save the review state for each artifact
- Support a change management process
- Provide traceability information for impact analysis

This work is especially interesting because with *Trac* one of the 3 compared tools uses a wiki and tickets to depict requirements.

2.7 Existing ReqM Tools in the Siemens Austria Context

Many requirements managers use simple standard software (text processing software, spreadsheet software) for requirements management. There are of course specialized tools. In this section, I describe the tools Requisite Pro (section 2.7.1) and Trac (section 2.7.2) commonly used at Siemens Austria. These tools are also the reference for comparison with the ReqM tool prototype in this work. I conclude this section with an overview of other well-known ReqM tools (section 2.7.3).

2.7.1 IBM Rational Requisiste Pro

Rational Requisite Pro [**IBM10**] is a dedicated requirements management tool. It is a desktop application, but a simple web interface has been added recently. One of its strengths is the integration with Microsoft Word, where requirements can be marked and are then automatically included in the database. Due to the software's evolution, some newer features like baselining have been added, but not been integrated seamlessly. Experts also frequently complain about the old-fashioned user interface. The integration with Requisite Pro works well for other Rational tools such as ClearQuest, but is very difficult for third-party or open-source tools.

2.7.2 Trac

Trac [**Tra10**] by Edgewall Software is an open-source web-based collaboration platform written in python. It allows tight integration with the Subversion [**Sub10**] version control system and is extensible via plug-ins. Two of Trac's main components are a wiki and a ticket system. For small projects, Trac is ideal as an allrounder solution because it can provide (together with subversion) version control, issue tracking, documentation, requirements and other services from one hand. Due to its plug-in architecture, Trac is very flexible and can easily be adapted to any project's specific requirements.

2.7.3 Well-known Tools Not Typically Used at Siemens Austria

Some well-known and frequently-used ReqM tools are Borland CaliberRM [**Bor10**], MKS Doors [**MKS10**], Sparx Enterprise Architect [**Spa10**], Geensoft Reqtify [**Gee10**]. This list was obtained from interviews with some ReqM experts (see section 3.3.1) and is neither ordered nor complete.

3 TreqPro – An Improved ReqM Concept

After giving an overview on the overall approach to an improved concept for an open-source requirements management tool in section 3.1, I describe the two research questions discussed in this work in section 3.2. The research questions are followed by a more detailed view of the approach in sections 3.3 (RQ1) and 3.4 (RQ2).

3.1 Overview of the Approach

This section gives an overview of the approach taken in this work to provide improved open-source requirements management compared to existing solutions (Figure 1).

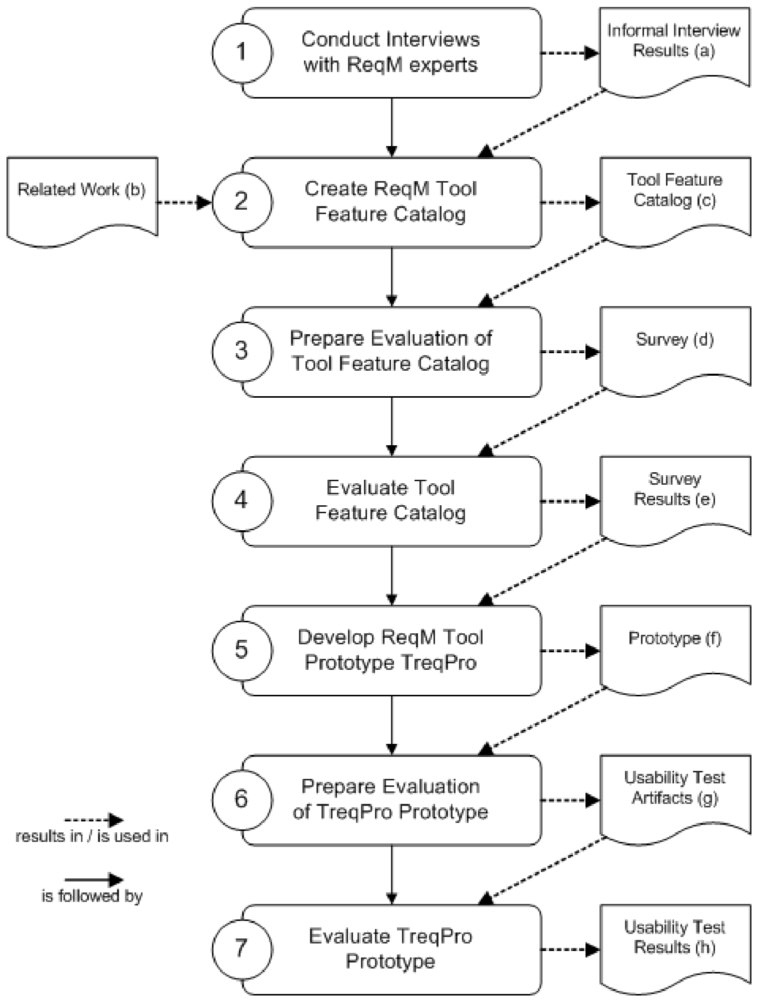

1 Conduct Interviews with ReqM experts
Informal Interview Results (a)
Related Work (b)
2 Create ReqM Tool Feature Catalog
Tool Feature Catalog (c)
3 Prepare Evaluation of Tool Feature Catalog
Survey (d)
4 Evaluate Tool Feature Catalog
Survey Results (e)
5 Develop ReqM Tool Prototype TreqPro
Prototype (f)
6 Prepare Evaluation of TreqPro Prototype
Usability Test Artifacts (g)
results in / is used in
is followed by
7 Evaluate TreqPro Prototype
Usability Test Results (h)

Figure 1: Simplified overview of approach to improved ReqM tool support

The following list gives an overview on each of the steps outlined in Figure 1. Steps (1) – (4) are outlined in detail in section 3.3. Steps (5) – (7) are discussed in section 3.4.

(1) I hold informal interviews with 5 ReqM experts at Siemens Austria in order to get an overview of important features and shortcomings of existing ReqM tools. The results of these interviews (a) are collected for the following steps.

(2) Using interview results (a) and existing work (b), I create an extensive catalog of ReqM tool features (c).

(3) For the evaluation of the catalog (c), I prepare a web survey (d) using the open-source tool *limesurvey*.

(4) The data yielded by the survey (d) is used to improve and validate the feature catalog (c). This is accomplished with a web survey. The results of the evaluation (e) encompass a list of features considered important by participants and an improved feature catalog.

(5) Using the improved feature catalog from (e), I develop a ReqM tool prototype (f).

(6) I prepare the evaluation of the prototype (f) which is realized as a usability test with ReqM experts. This encompasses the creation of a sample project with valid and consistent test data, the deployment of the prototype (f) to a virtual machine, and the creation of extensive survey response sheets. All these artifacts (g) are used in the following steps.

(7) 8 participants take part in the tests, where each test uses a set of artifacts from (g). I record the results (h) consisting of quantitative and qualitative parameters.

3.2 Research Questions

This section outlines the actual research questions RQ1 (section 3.2.1) and RQ2 (section 3.2.2) posed in this work. The approach taken for each of these questions is described in more detail in sections 3.3 (for RQ1) and 3.4 (for RQ2).

3.2.1 RQ1: Which features are essential for ReqM tool support?

On one hand, many requirements managers (especially in smaller projects) use standard software for elicitation and management of requirements such as text processing software and spreadsheet software. On the other hand, there are numerous existing ReqM tools with a variety of different features.

It is not obvious, which features should be included in a good ReqM tool. This research question therefore aims at discovering the features considered most important by requirements managers. It is vital to answer this question to affirm the relevance of RQ2, and to elicitate features that should be included in a first prototype.

3.2.2 RQ2: How can requirements versioning, traceability and tool integration be improved with TreqPro (compared to Siemens Standard ReqM tools)?

I have identified three major shortcomings in existing ReqM tools (see section 1.2.5):

Requirements traceability (see section 1.2.5.1) is typically limited in two ways: (1) Traceability can only be established among requirements managed inside the tool, but not towards design documents, source code, test cases, etc. (2) Traces cannot have multiple user-defined types.

Requirements versioning (see section 1.2.5.2) is typically implemented in a simple, linear way that does not allow switching back to old versions of a requirement, creating baselines of the whole project, or comparing baselines and artifact versions in a clear and easy way.

ReqM tool integration (see section 1.2.5.3) is typically possible between tools of the same manufacturer, but hard to achieve with third-party tools. For example, it is hard to link requirements to source code with some ReqM tools. Data is usually spread across multiple repositories, which creates redundancies and complicates usage of the data in other tools.

In this work, I investigate, how these shortcomings of ReqM tools can be addressed by developing a general concept of addressing the shortcomings and implementing that concept in a prototype.

3.3 Elicitation and Validation of a ReqM Tool Feature Catalog

RQ1 addresses features necessary for ReqM tool support. In this part of the work, I create a comprehensive catalog of ReqM tool features from multiple sources.

3.3.1 Sources of ReqM Tool Features

As outlined in section 2.2, I use 2 primary sources of ReqM tool features: Tool databases and existing literature. To supplement the knowledge gained from these sources I hold several interviews with ReqM experts and project managers at Siemens Austria to elicit up-to-date information about requirements for ReqM tools. The interviews are held in an informal way and all information is recorded in written form.

3.3.2 Abstraction of Requirements to Artifacts

I collect and sort the requirements for ReqM tools acquired in the previous step. Furthermore, I introduce an abstraction from requirements to *artifacts* in order to allow the concepts introduced for requirements to be applied to similar domains like test management.

Each artifact can have one or more *attributes.* An attribute can contain any data that is associated with the artifact. Artifacts can be connected to each other by *relations*, as shown in Figure 2.

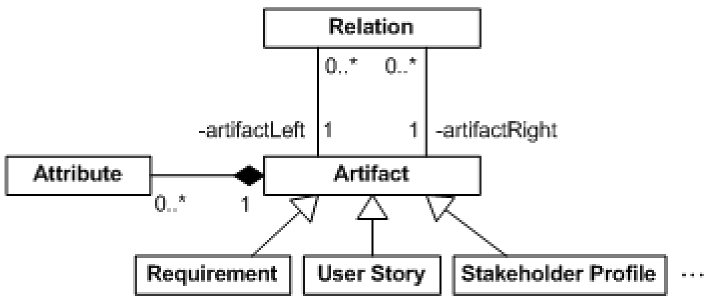


Figure 2: Abstraction of requirements to artifacts with attributes and relations

Both attributes and relations have types. Depending on the implementation, this allows semantic tracing, enforcement of certain relations, and better structuring of requirements.

3.3.3 Elicitation and Validation of the Catalog.

The full feature catalog is included as Appendix A. The catalog will be evaluated using a web survey, see section 4.1.

3.4 Development of the TreqPro Prototype

RQ2 deals with a concept to improve ReqM compared to existing tool solutions, especially in the field of the 3 shortcomings described in section 3.2.2. In this part of the work, I describe such a concept and create an open-source ReqM tool prototype which implements that concept.

3.4.1 Concept for Improved ReqM Tool Support

The following sections describe how I use existing open-source tools to create an open-source ReqM tool as well as each major shortcoming identified before (see section 3.2.2) and how I plan to address it.

3.4.1.1 Building an Open-Source Solution

There are already many commercial, proprietary ReqM tools. These tools typically have potentially high license and training costs, and their use is sophisticated and hard to learn. Proprietary tools only offer limited integration (especially with third-party tools) and limited extensibility. Most FLOSS tools (further referred to as ‘open-source tools’, see section 1.2.1) are not subject to these limitations. Open-source tools have therefore become increasingly popular in software development in the past few years (e. g. *Subversion*, *Bugzilla*). At the moment, there are no usable open-source ReqM tools. The most notable open-source ReqM tool is the open-source requirements management tool (OSRMT) [**Ost10**]. OSRMT has not been maintained since 2007. Considering the advantages of open-source software, I have decided to create a web-based open-source solution as a base for the improved ReqM concept. I use the revision control system *Subversion* [**Sub10**], the collaboration platform *Trac* [**Tra10**] and a few existing Trac plug-ins to build the new ReqM tool prototype ***TreqPro***.

3.4.1.2 Traceability

Traceability helps avoid redundancies in requirements and “over-engineering” in design and allows anticipating the effects of requirements changes. It is important to maintain traceability throughout all phases of development [**Ram97**]. This is difficult to achieve if traceability has to be established manually. ReqM tools should therefore try to facilitate establishing traceability as much as possible.

The generation of traceability is a critical factor, because the quality of traceability decreases with increasing effort for the user, as explained by Spanoudakis et al. [**Spa05**]. According to the authors, manual generation of traceability requires much effort on the user side and is error-prone. Techniques for full-automatic generation are still in development and difficult to implement. In the prototype, I use the semi-automatic approach of **process-driven generation of traceability** described by Spanoudakis et al.

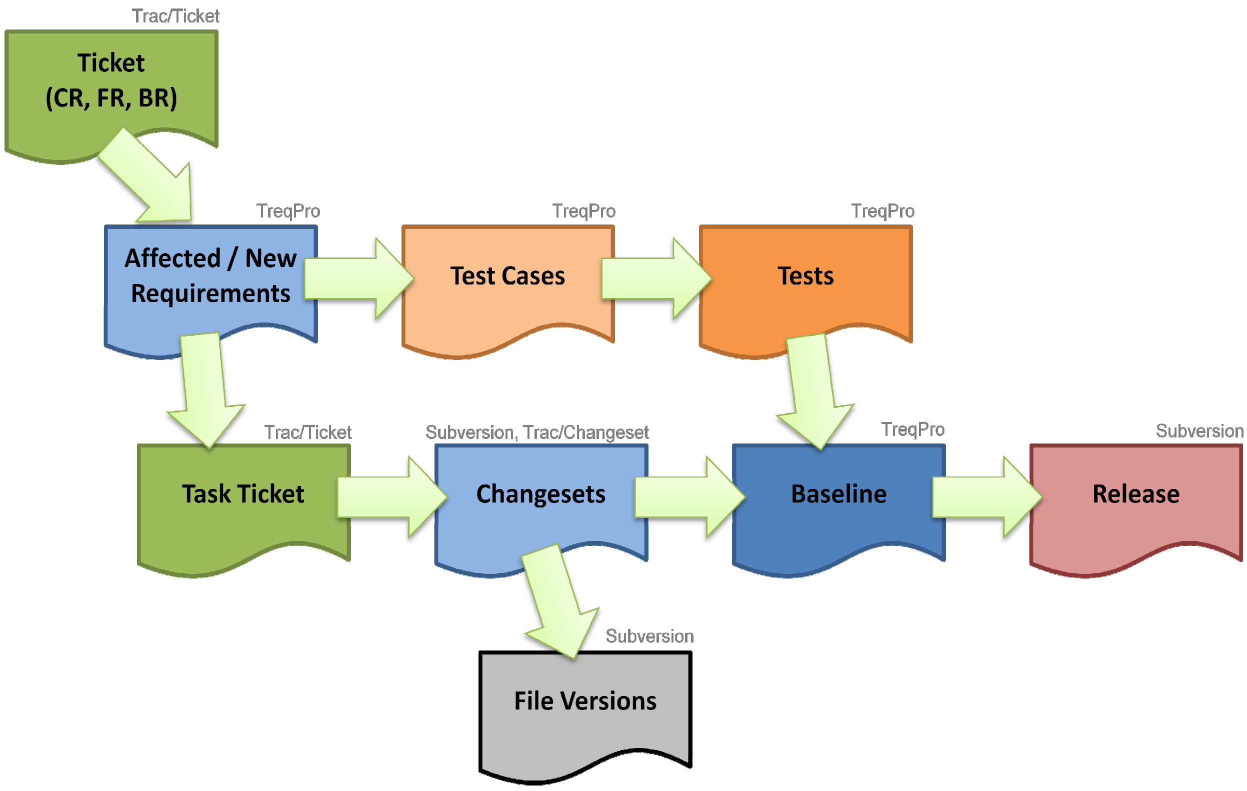


Figure 3: Process-driven generation of traceability [Spa05] **(figure derived from original by** [Rei09]**)**

Inside the prototype, traceability with multiple named relation types, further referred to as "semantic tracing", has been implemented to different extent for practical reasons as described in the following paragraphs.

TreqPro Artifacts and TreqPro Artifacts – Full Semantic Tracing

TreqPro artifacts can be linked to each other using relations. Each relation has a named relation type. Relations can be created using an artifact browser from any artifact's view or using a traceability matrix. I have developed an experimental graphical relation browser that can display an overview of multiple related artifacts for quick navigation.

Tickets and TreqPro Artifacts – Limited Semantic Tracing

Tickets and other TreqPro artifacts (such as requirements) can be associated with each other from the ticket view as well as from the artifact view using a simple selection list. This type of association is not named, but basic semantics are provided by displaying of the artifact/ticket type on the opposite part of the association. This implementation is not optimal because the kind of relation differs from the one used between artifacts. In further work, this should be improved to provide only one kind of relation.

Task Tickets and Changesets – Limited Semantic Tracing

When committing a changeset to a subversion repository, a hook script associates the change with the respective tickets if mentioned in the commit message. It is possible to add additional validation or action in the hook script, e. g. to only allow commits to open tickets, or to automatically close tickets. This implementation is not optimal because the kind of relation differs from the one used between artifacts. In further work, this should be improved to provide only one kind of relation.

3.4.1.3 Versioning

Versioning vs. History

Some existing ReqM tools record the history of changes without the possibility to view a full diff or restore an old version. Viewing the exact changes allows users a much better understanding of the evolution of a requirement. Versioning can be implemented by simply storing each version or by storing the first version plus differential information for each change.

Linear Versioning vs. Extended Versioning with Branching

Many software systems which implement some kind of versioning only offer linear versioning, i. e. each version has zero or one predecessor and zero or one successor. Linear versioning has the advantage that it is easy to use and easy to implement. A major drawback is that after restoring an old version, all newer versions have to be discarded as soon as changes are made to that old version – otherwise that version would have multiple successors. By allowing more than one successor, this disadvantage can be removed. Linear versioning is a subset of the extended versioning, and it is not necessary to use branching if the user wants to use simple linear versioning. Each version still has zero or one predecessor in extended versioning.

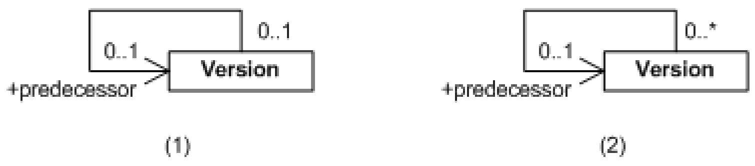


Figure 4: Linear versioning (1) vs. extended versioning with branching (2)

Project Versioning and Baselining

The state of a requirements project while working on requirements is called a “working version”. At certain points in the project’s lifecycle, it is important to make a snapshot of all requirements to have a basis for negotiations or presentation of a certain state of the project. Such a snapshot is called a baseline.

A project version is technically a list of artifact versions. While a working version can be modified, a baseline is a read-only snapshot of the contained artifact versions. A baseline cannot be changed after creation. It is therefore important to guarantee that the contents of a baseline cannot be altered or deleted:

- Each change of an artifact creates a new version of that artifact.
- Each baseline contains zero or one versions of each artifact in the project.
- The database prevents deletion of any version that is used in a baseline.

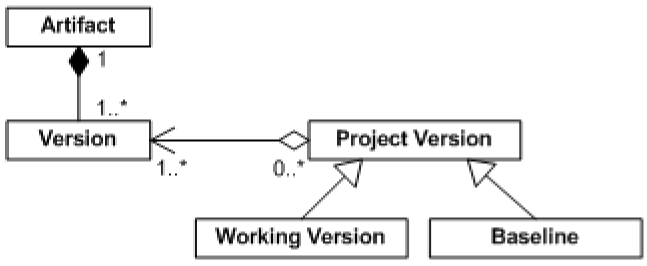


Figure 5: A project version can be a working version (modifiable) or a baseline (read-only)

3.4.1.4 Addressing Limited Integration

The TreqPro prototype tries to improve on existing ReqM tool concepts by integrating all functionality usually distributed among multiple tools into one platform and by using only two repositories (a relational database and a version control repository). This concept removes the need for interfaces that are hard to maintain and facilitates data exchange between different tools.

TreqPro has been developed as a plug-in for the Trac platform, which itself integrates well with the Subversion version control system. Trac plug-ins can offer functionality to each other and share the basic services of the collaboration platform, such as wiki, tickets, search, repository browsing, etc. All Trac plug-ins can access the whole database and thus have access to all available data.

The Trac platform itself already provides many basic features necessary for ReqM tools. The TreqPro plug-in adds a data model and special functionality especially designed for requirements management as well as a highly specialized user-interface for editing and managing requirements. It is easy to add more functionality to the platform if required by developing additional plug-ins.

3.4.2 Essential Use Cases

The essential use cases cover basic ReqM functionality as well as functionality that address the shortcomings of existing ReqM software. I have specified the following essential use cases.

3.4.2.1 Select a Requirements Project

Use Case	EU-1 Select a requirements project
Actors	User
Description	A user wants to use a specific requirements model. He selects a requirements project that will be loaded into the application. Requirements projects can be created, cloned, deleted and baselined. A baseline can become a work version (e. g. indicated by a '*'), for which it has to be cloned first.
Normal Course	1. The user indicates that he would like to select a requirements project. 2. The system displays a list of all requirements projects and their baselines. 3. The user selects one of the projects and confirms the action. 4. The system loads the selected project 5. The system displays the first artifact.
Preconditions	• The user is logged in.
Postconditions	• A requirements project is selected.

Table 1: EU-1 Select a requirements project

3.4.2.2 Create a New Requirement

Use Case	EU-2 Create a new requirement
Actors	User
Description	A user wants to enter a new requirement.
Normal Course	1. The user indicates that he would like to create a new requirement. 2. The user indicates the desired position of the new artifact in the requirements project. 3. The system displays a form for the new requirement that contains all the fields that have been defined for the artifact type 'requirement'. 4. The user fills in the form and confirms the action. 5. The system validates all the submitted values. 6. The system creates the artifact as specified. 7. The system creates relations to reflect the position of the artifact as

	necessary. 8. The system displays the new artifact.
Preconditions	• The user is logged in. • A requirements project is selected.
Postconditions	• A requirements project is selected.

Table 2: EU-2 Create a new requirement

3.4.2.3 Change a Requirement

Use Case	EU-3 Change a requirement
Actors	User
Description	A user wants to change an existing requirement after a customer has issued a change request. The change request must be linked to the new version of the requirement.
Normal Course	• The user indicates that he would like to change a certain requirement. • The user selects the change request that is associated with the change. • The system displays a form for the existing requirement that contains all the fields that have been defined for the artifact type 'requirement'. • The user makes his changes in the form and confirms the action. • The system validates all the submitted values and checks the associated change request. • The system creates a new version of the requirement artifact as specified. • The system closes the change request and/or changes fields in the change request as appropriate. • The system creates/modifies/deletes relations to reflect the position/state of the artifact as necessary. • The system sets all relations of the changed artifact to 'suspect'. • The system displays the changed artifact.
Triggers	• A customer issues a change request.
Preconditions	• The user is logged in.

Postconditions	• A new version of the requirement has been created.
Invariants	• The existing version of the requirement is never modified.

Table 3: EU-3 Change a requirement

3.4.2.4 Establish Traceability for an Existing Requirement

Use Case	EU-4 Establish Traceability for an Existing Requirement
Actors	User
Description	A user wants to link an existing requirement to a user story (backward traceability) and to the source code (forward traceability). Other frequently used link targets would be design documents and test cases.
Normal Course	1. The user indicates that he would like to link the current artifact to the user story using the "derived_from" relation. 2. The system validates all the submitted values. 3. Thes system creates a new bidirectional relation souce_of / derived_from between a new version of the user story and a new version of the selected requirement. 4. The system displays the new version of the requirement.
Triggers	• A new requirement has been entered.
Preconditions	• The user is logged in. • The user has entered a new requirement. • The new requirement has been selected and is displayed. • There is a user story the new requirement is derived from.
Postconditions	• A new bidirectional relation to the user story has been created. • A new relation to the source code has been created.
Invariants	• The relations and attributes of the existing version of the user story are not modified. • The relations and attributes of the existing version of the requirement are not modified.

Table 4: EU-4 Establish traceability for an existing requirement

3.4.2.5 Query, Filter and Sort Requirements According to Certain Criteria

Use Case	EU-5 Query, filter and sort requirements according to certain criteria
Actors	User
Description	A user wants to find all requirements derived from user stories by a certain user.
Normal Course	1. The user indicates that he would like to issue a query. 2. The user enters the query using SQL and assisted input to find all requirements which are linked to user stories which are linked to user profiles where the fist name is 'Frank'. 3. The system checks and filters the SQL code and then retrieves the desired artifacts. 4. The system displays the results in a formatted, sortable and clickable list. 5. The user indicates that he would like to sort the list by date. 6. The system displays the list sorted by date. 7. The user clicks the first requirement. 8. The system displays the requirement.
Preconditions	• The user is logged in.

Table 5: EU-5 Query, filter and sort requirements according to certain criteria

3.4.2.6 Add a new Field to an Artifact Type

Use Case	EU-6 Add a new field to an artifact type
Actors	Project Administrator
Description	A user wants to add the new field 'cost' to the the artifact type 'requirement'.
Normal Course	1. The user indicates that he would like to modify a certain requirements model. 2. The system loads the specified requirements model and displays all artifact types. 3. The user indicates that he would like to modify the artifact type 'requirement'. 4. The system loads the specified artifact type and displays all

	attribute types. 5. The user indicates that he would like to add a new mandatory field with the data type 'float', the name 'cost', and the default value '0.0'. 6. The system creates the new mandatory attribute type 'cost' with data type 'float' and default value '0.0'. 7. The system adds the new attribute type 'cost' to the allowed attribute types of the artifact 'requirement'. 8. The system loads the artifact type 'requirement' and displays all attribute types (including the new one) again.
Preconditions	• The administrator is logged in. • There is at least one requirements project containing meaningful data. • The artifact type 'requirement' exists.
Postconditions	• A new attribute type 'cost' has been created and added to the artifact type 'requirement'.

Table 6: EU-6 Add a new field to an artifact type

3.4.2.7 Generate a Requirements Document

Use Case	EU-7 Generate a requirements document
Actors	User
Description	A user wants to generate a requirements specification document out of existing artifacts. The project is automatically baselined.
Normal Course	1. The user indicates that he would like to create a requirements specification document. 2. The system informs the user that all artifacts in the project will be baselined. 3. The user confirms the action. 4. The system creates a requirements specification document and adds it to the project. 5. The system creates a baseline that includes all artifacts in the project. 6. The system changes the view to display the newly created requirements specification document.

Preconditions	• The user is logged in. • A requirements project is selected. • The project contains artifacts.
Postconditions	• A requirements specification document has been created. • The requirements specification document is part of a new baseline that has been created.

Table 7: EU-7 Generate a requirements document

3.4.2.8 Compare Two Project Versions

Use Case	EU-8 Compare two project versions
Actors	User
Description	A user wants to get aan overview about two project tversions (which are either read-only baselines or read-write working versions).
Normal Course	1. The user indicates that he would like to compare baselines. 2. The system displays a list of all available baselines for the current project. 3. The user selects two baselines. 4. The system displays all versions in the two baselines in a way so the user can easily recognize differences.
Preconditions	• The user is logged in. • A requirements project is selected. • The project contains two or more baselines which contains further data.
Postconditions	• A requirements project is selected.

Table 8: EU-8 Compare two project versions

3.4.2.9 Compare Two Artifact Versions

Use Case	EU-9 Compare two artifact versions
Actors	User
Description	A user wants to see all differences between two versions of the same artifact.
Normal Course	1. The user indicates that he would like to see the history of an artifact.

	2. The system displays all available artifact versions in a tree. 3. The user selects two versions to compare. 4. The system displays a diff of the two versions.
Preconditions	• The user is logged in. • A requirements project is selected. • The project contains artifacts.

Table 9: EU-9 Compare two artifact versions

3.4.2.10 Graphical Navigation

Use Case	EU-10 Graphical navigation
Actors	User
Description	A user wants to navigate to an artifact that is connected to the current artifact over a distance of at least 2 relations.
Normal Course	1. The user indicates that he would like to see artifacts related to the current artifact. 2. The system displays a graphical view of all related artifacts. 3. The user selects the new artifact. 4. The system loads, selects and displays the given artifact.
Preconditions	• The user is logged in. • An artifact is selected and displayed.
Postconditions	• The new artifact, which has a distance of at least two relations to the first artifact, is selected and displayed.

Table 10: EU-10 Graphical navigation

3.4.3 User Interface Concept

The TreqPro user interface is designed after the use cases. The following section gives a schematic overview of the user interface with a focus on the main screen (Figure 6). The description is supplemented by screenshots of the most important interfaces.

Project Browser

The project browser displays a list of all projects and all project versions for the selected project. It allows comparing two specific project versions to each other and highlights the differences. It also allows creating baselines and selecting specific project versions as well as changing the selected project.

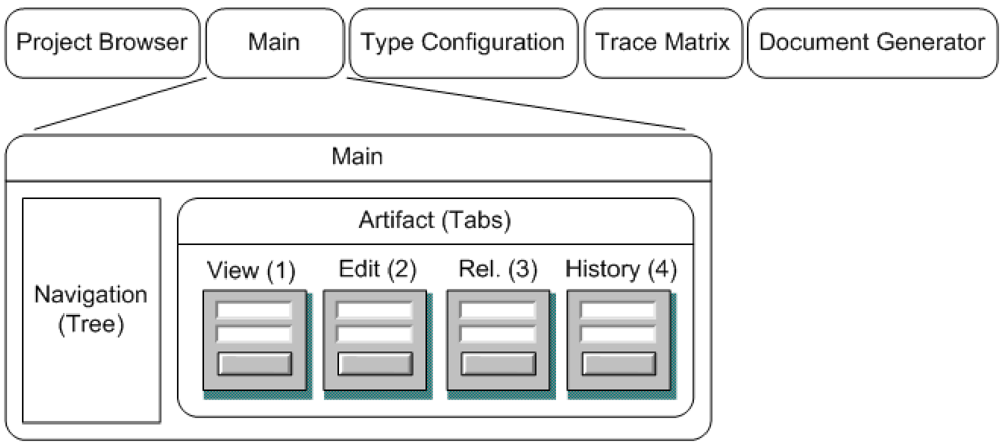
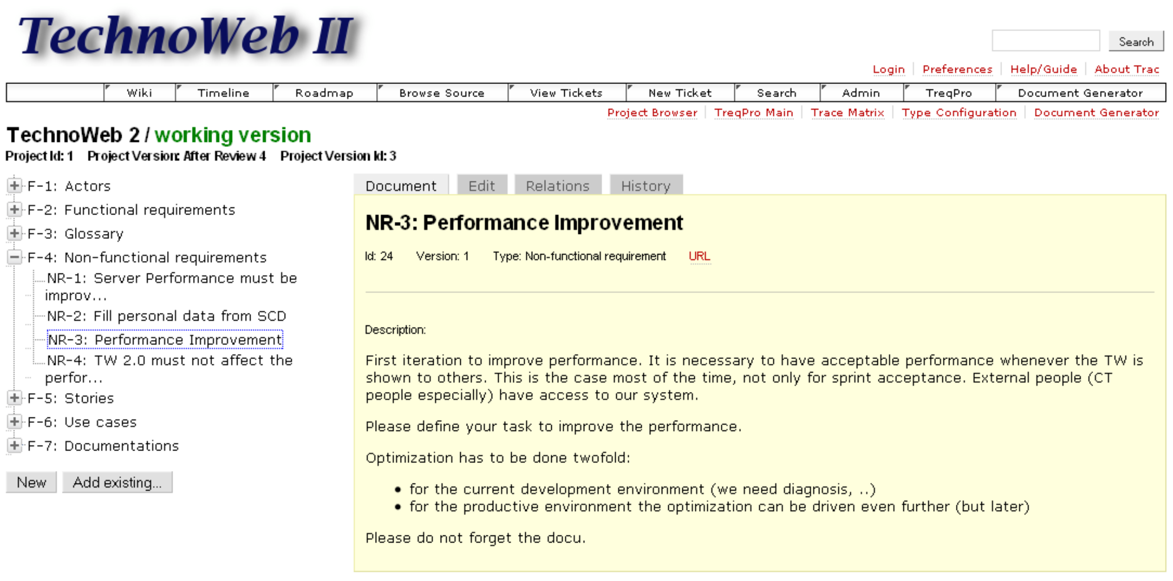


Figure 6: User interface concept showing partition of the main screen that allows viewing (1), editing (2), managing relations (3), and accessing the history (4) of the currently selected artefact.

Main

The main view contains a tree view as navigation next to a tab control with the following tabs: The view tab (1) is a textual display consisting of all attributes of an artifact (Figure 7). The edit tab (2) contains editing fields for all artifacts and also allows wiki syntax editing for fields with appropriate type (Figure 8). The relations tab (3) shows all relations from this artifact to any other artifact in a textual and graphical, experimental way (Figure 10). The history tab (4) shows all versions of the selected artifact and allows comparing specific versions and highlighting the differences or selecting any existing artifact version into the current project version (for reverting or branching, see Figure 9).


TechnoWeb II
Search
Login | Preferences | Help/Guide | About Trac
Wiki | Timeline | Roadmap | Browse Source | View Tickets | New Ticket | Search | Admin | TreqPro | Document Generator
Project Browser | TreqPro Main | Trace Matrix | Type Configuration | Document Generator
Document | Edit | Relations | History
New | Add existing...

TechnoWeb 2 / working version

Project Id: 1 Project Version: After Review 4 Project Version Id: 3

F-1: Actors
F-2: Functional requirements
F-3: Glossary
F-4: Non-functional requirements
NR-1: Server Performance must be improv...
NR-2: Fill personal data from SCD
NR-3: Performance Improvement
NR-4: TW 2.0 must not affect the perfor...
F-5: Stories
F-6: Use cases
F-7: Documentations

NR-3: Performance Improvement

Id: 24 Version: 1 Type: Non-functional requirement URL

Description:

First iteration to improve performance. It is necessary to have acceptable performance whenever the TW is shown to others. This is the case most of the time, not only for sprint acceptance. External people (CT people especially) have access to our system.

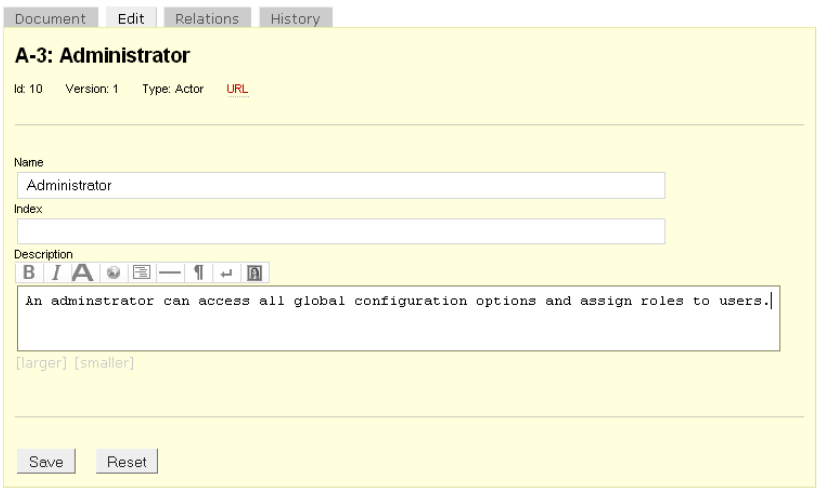
Please define your task to improve the performance.

Optimization has to be done twofold:

- for the current development environment (we need diagnosis, ..)
- for the productive environment the optimization can be driven even further (but later)

Please do not forget the docu.

Figure 7: The view of a selected artifact displays all its attributes.


Document | Edit | Relations | History

A-3: Administrator

Id: 10 Version: 1 Type: Actor URL

Name

Administrator

Index

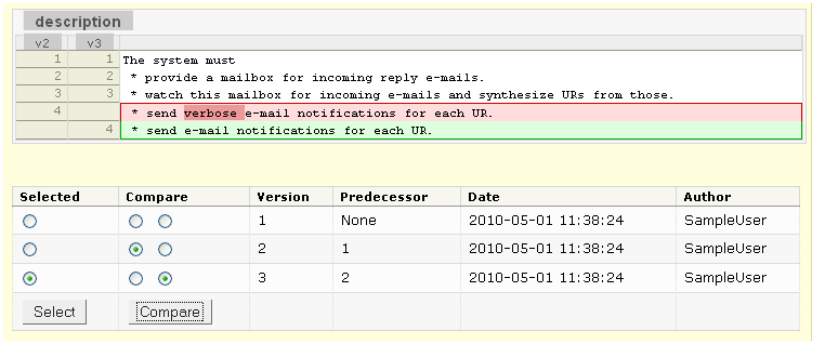
Description

```
An adminstrator can access all global configuration options and assign roles to users.
```

[larger] [smaller]

Save | Reset

Figure 8: The application allows direct editing of each attribute and multi-line editing with wiki syntax.


description

v2	v3	
1	1	The system must
2	2	* provide a mailbox for incoming reply e-mails.
3	3	* watch this mailbox for incoming e-mails and synthesize URs from those.
4		* send verbose e-mail notifications for each UR.
	4	* send e-mail notifications for each UR.

Selected	Compare	Version	Predecessor	Date	Author
○	○ ○	1	None	2010-05-01 11:38:24	SampleUser
○	● ○	2	1	2010-05-01 11:38:24	SampleUser
●	○ ●	3	2	2010-05-01 11:38:24	SampleUser
Select	Compare				

Figure 9: The exact changes between two arbitrary versions are clearly highlighted.

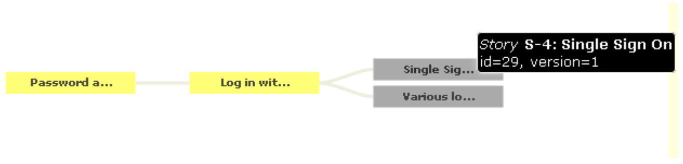


Figure 10: The experimental graphical navigation allows viewing relations across multiple artifacts.

Trace Matrix

The trace matrix is a table showing all relations of the selected type and allowing for an overview and quick setting of multiple relations. Large trace matrices can be prevented using filters.

Type Configuration

The relation type configuration consists of names for both relation directions, relation modifiers like 'composite', and allowed artifact types on each side of the relation. The artifact type configuration allows managing attribute type details (name, data type, control type, order) for each artifact type.

Document Generator

The document generator implemented in the prototype has a very simple user interface with only two buttons, one for CSV export and one for PDF export.

3.4.4 Software Architecture

This section describes architectural aspects of the TreqPro prototype.

3.4.4.1 Integration of the Prototype into the Trac Platform

The prototype is implemented as a plug-in to the open-source collaboration platform Trac [**Tra10**]. It is programmed in Python [**Pyt10**] and installed into a Trac environment as a python egg. A PostgreSQL relational database and a Subversion version control repository provide persistence. See Figure 11 for an overview of the platform.

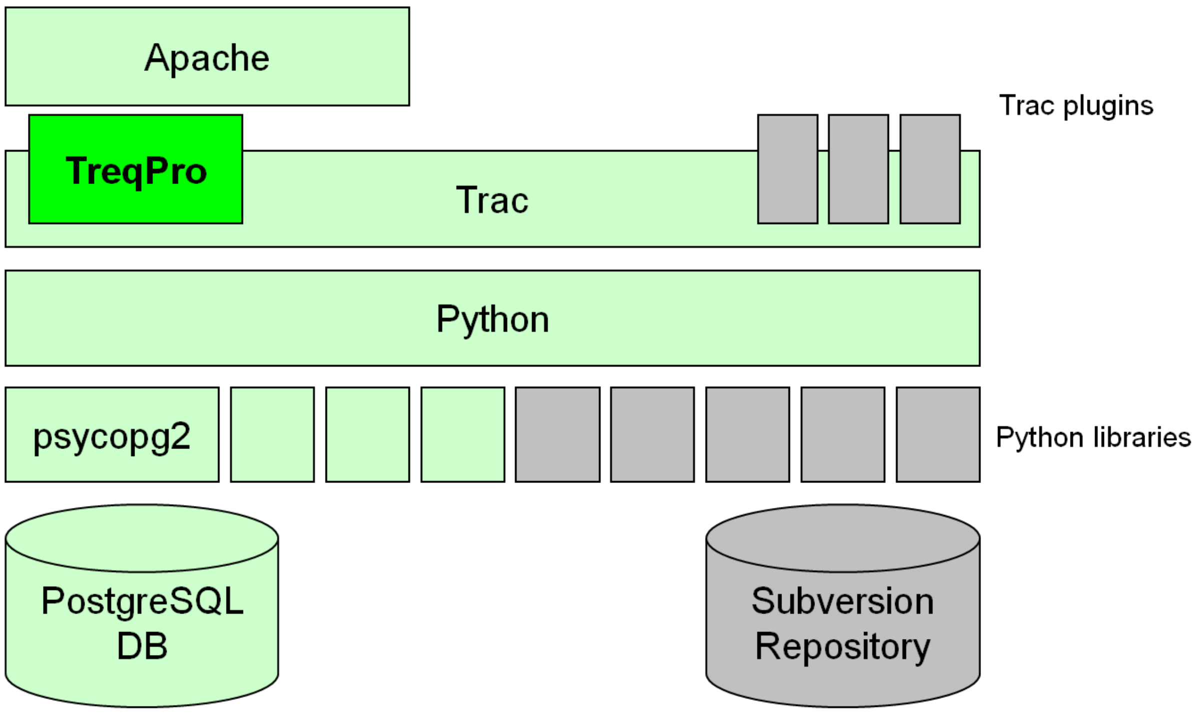


Figure 11: TreqPro software stack.

3.4.4.2 Internal Architecture

TreqPro is a distributed application. Its architecture is therefore described separately for the client and the server side of the application (Figure 12).

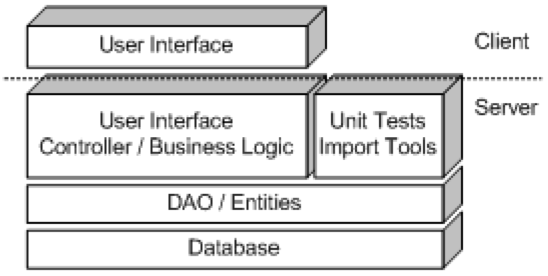


Figure 12: Architectural Overview

User Interface (Client)

TreqPro is a web application that is accessible via web browser. It has been tested most extensively with Mozilla Firefox 2 and Mozilla Firefox 3.

TreqPro contains JavaScript code that is executed by the web browser on the client side. The JavaScript code is used for creating the tree in the main view, validating input forms and many small improvements of the user experience (e. g. text box resizing). The graphical presentation of relations is also realized using JavaScript.

The following FLOSS JavaScript libraries are used by TreqPro:

- Yahoo UI Library (2.7.0b) for tabs and the tree view
- JQuery (1.7.2)
- JavaScript InfoVis Toolkit (1.1.3)

User Interface / Business Logic (Server)

This part of the application is Python code. TreqPro is built on the Trac platform and therefore implements several Trac interfaces. Its user interface is highly integrated with Trac. There is one request handler (implementing Trac's *RequestHandler* interface) for each main part of the application and one for AJAX requests. Each request handler handles the server-side part of the user interface and contains the business logic. Most of the user interface is defined in Genshi XML templates which are provided with data by the request handlers. Table 11 gives an overview on the modules of the user interface.

ajax.py	AJAX requests, main screen
ajax_projects.py	AJAX requests, project browser
artifact_browser.py	artifact browser
diff.py	project diffs
main.py	main screen
matrix.py	trace matrix
projects.py	project browser
reqdocgen.py	document generator
types.py	type configuration

Table 11: Modules of the user interface

DAO / Entities (Server)

The entity layer (Figure 13) maps artifacts, relations, projects, project versions, artifact types and relation types as well as some helper objects. The entities are designed around the Active Record pattern [**Fow03**].

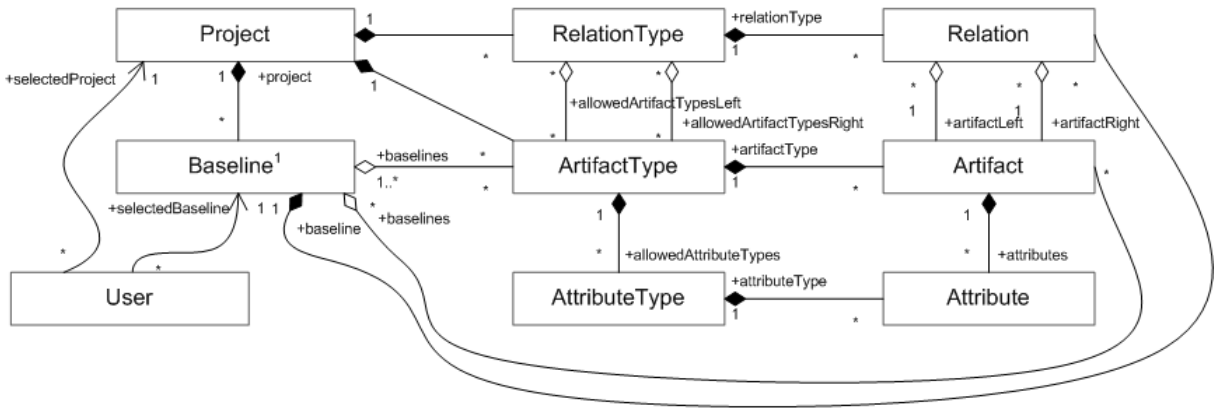


Figure 13: Excerpt of the TreqPro object model

Unit Tests / Import Tools (Server)

To assess the quality of the DAO layer, a set of unit tests is created that can be run from the IDE using *PyUnit*. The unit tests are extremely important because adding new features to the DAO layer can cause already tested code to fail, and manual regression testing would be sophisticated.

A script to import artifact data from a CSV file is also available. This is highly useful for testing purposes.

Database (Server)

The PostgreSQL database (Figure 14) contains constraints and stored procedures to ensure the consistency of the data (especially for versioning and baselines). The plug-in shares a PostgreSQL database with other Trac plug-ins. Other database management systems are currently not supported due to the use of non-portable stored procedures.

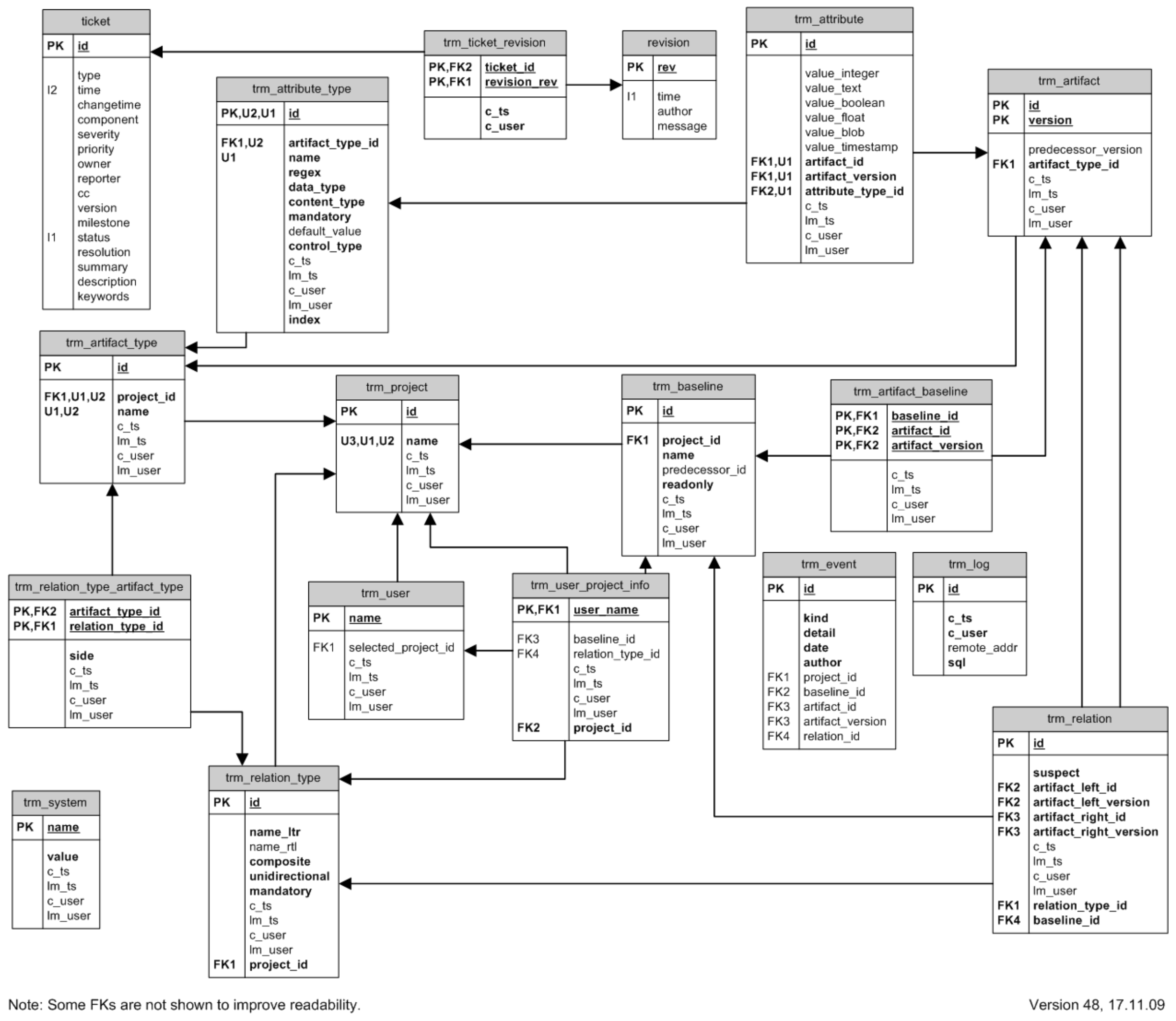


Figure 14: Excerpt of the TreqPro database schema

3.4.4.3 TreqPro's Dependencies

The prototype uses the Document Generator plug-in by Alexander Wagner to generate a Requirements Document.

To set up the development environment on a MS Windows XP SP3 system, at least the following software packages are required:

- Python 2.5.4
    - eGenix-mx-base 3.1.2
    - psycopg2 2.0.10-pg8.3.7
    - setuptools 0.6c9
    - svn-python 1.6.6
    - Genshi 0.5.1
- Postgresql 8.3.7-1
- JRE 6 Update 20
- Eclipse SDK 3.5
- Trac 0.11.4
- Subversion 1.6.6
- Adobe Reader 9.3.2
- Mozilla Firefox 3.5.10
- TortoiseSVN 1.6.7

3.4.5 Feature Matrix of all 3 Compared Tools

The feature matrix (Table 12) compares important features in the 3 evaluated tools (++ implemented, + partially implemented, – not implemented). The features are taken from the catalog (see section 3.3).

	Group A Requisite Pro	Group B Trac	Group C TreqPro
web interface	–	++	++
full project template configuration (artifacts and relations)	+	–	++
semantic tracing (relation types)	–	–	++
suspect tracing	++	–	–
basic versioning	++	++	++
baselining	++	–	++
branching	–	–	++
comparison of baselines	+	–	++
comparison of artifact versions	+	–	++
traceability across tool borders	+	+	++
traceability matrix	++	–	++
wysiwyg editing	+	+	+
report generation	+	–	++
IDE integration	–	–	–
configuration management integration	–	++	++
test management integration	–	+	+
graphical navigation	–	–	+
workflow support for requirements	–	–	–

Table 12: Feature matrix of the 3 tools compared in the evaluation

4 Evaluation

To answer the research questions, I carry out an evaluation consisting of 2 parts. Each part deals with one of the research questions. The evaluation is designed as an empiric study and aligned to published best practices [**Vis02**].

RQ1: What features are essential for ReqM tool support?

This part evaluates the completeness of the feature catalog for ReqM tool support. This research question will be addressed with a survey. This part is described in **section 4.1**.

RQ2: How can requirements versioning, traceability and tool integration be improved with TreqPro (compared to Siemens Standard ReqM tools)?

This part evaluates with a usability test how well the tool integration concept implemented in the TreqPro prototype addresses the shortcomings compared to an existing commercial product and an existing open-source solution, both commonly used at Siemens Austria. This part is described in **section 4.2**.

4.1 Evaluation of the Feature Catalog

This part of the evaluation will test, to which extent essential requirements for ReqM tools are covered in the feature catalog that I have created.

This part can be split up into the following questions:

1. How well does the feature catalog cover essential ReqM tool features?
2. What important ReqM tool features are not yet in the feature catalog?

The feature catalog for ReqM tools created in this work will be compared to the results of a survey among ReqM experts. In this survey, each of the experts will be asked to provide a list of the ReqM tool features that he or she considers most important.

4.1.1 Factors Affecting this Part of the Evaluation

The following factors affect this part of the evaluation (Table 13).

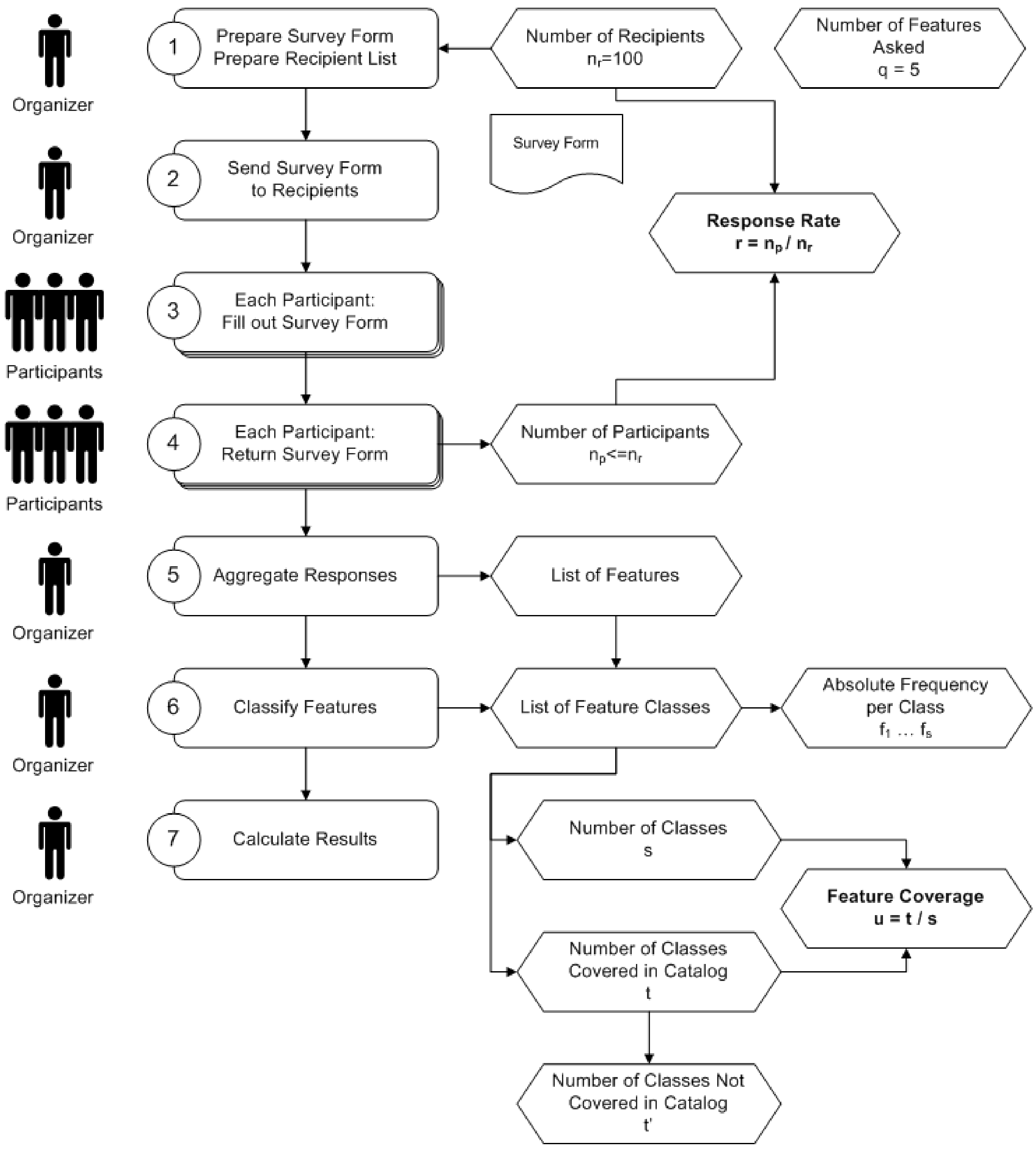
n_r	**Number of recipients:** The number of recipients of the survey invitation e-mail
n_p	**Number of participants:** The number of recipients actually returning a correctly filled-out survey form
q	**Number of features asked:** The number of features that each participant is asked to provide
r	**Response rate:** Number of participants in relation to the number of recipients ($r = n_p / n_r$)
	List of Features: All features provided by participants, sorted
	List of Feature Classes: A classification of all the features provided by participants
s	**Number of feature classes**
$f_1 \dots f_s$	**Absolute frequency per class:** Number of features assigned to each class
t	**Number of feature classes covered in the catalog**
t'	**Number of feature classes not covered in the catalog**
u	**Feature Coverage:** Number of feature classes covered in the catalog in relation to the total number of feature classes ($u = t / s$)

Table 13: Factors affecting the evaluation of the feature catalog

The detailed interaction of these factors is displayed in Figure 15 in the following section.

4.1.2 Evaluation Method for the Feature Catalog

This section outlines the process of this part of the evaluation in detail (see Figure 15).



Organizer
1 Prepare Survey Form Prepare Recipient List
Number of Recipients n_r=100
Number of Features Asked q = 5
Survey Form
Organizer
2 Send Survey Form to Recipients
Response Rate $r = n_p / n_r$
Participants
3 Each Participant: Fill out Survey Form
Participants
4 Each Participant: Return Survey Form
Number of Participants n_p<=n_r
Organizer
5 Aggregate Responses
List of Features
Organizer
6 Classify Features
List of Feature Classes
Absolute Frequency per Class $f_1 \dots f_s$
Organizer
7 Calculate Results
Number of Classes s
Feature Coverage u = t / s
Number of Classes Covered in Catalog t
Number of Classes Not Covered in Catalog t'

Figure 15: The features listed in survey responses are classified and the catalog's coverage of these features is calculated.

The feature catalog is evaluated inside Siemens Austria using a web survey.

4.1.2.1 Preparation of Evaluation

1. The organizer prepares a web survey form and an e-mail invitation asking participants to list a certain number of tool features that are essential for ReqM tools in their opinion:
   - The survey form has to outline the advantages of the catalog and its possible use for comparing future ReqM tools.
   - The short time necessary for taking part in the survey should also be mentioned.
   - The participants are asked to answer freely so they can give unexpected answers.
   - The participants should give general answers that do not include specific technologies.
   - There will be a small reward to encourage the recipients to participate: The first 5 participants receive a copy of the feature catalog.

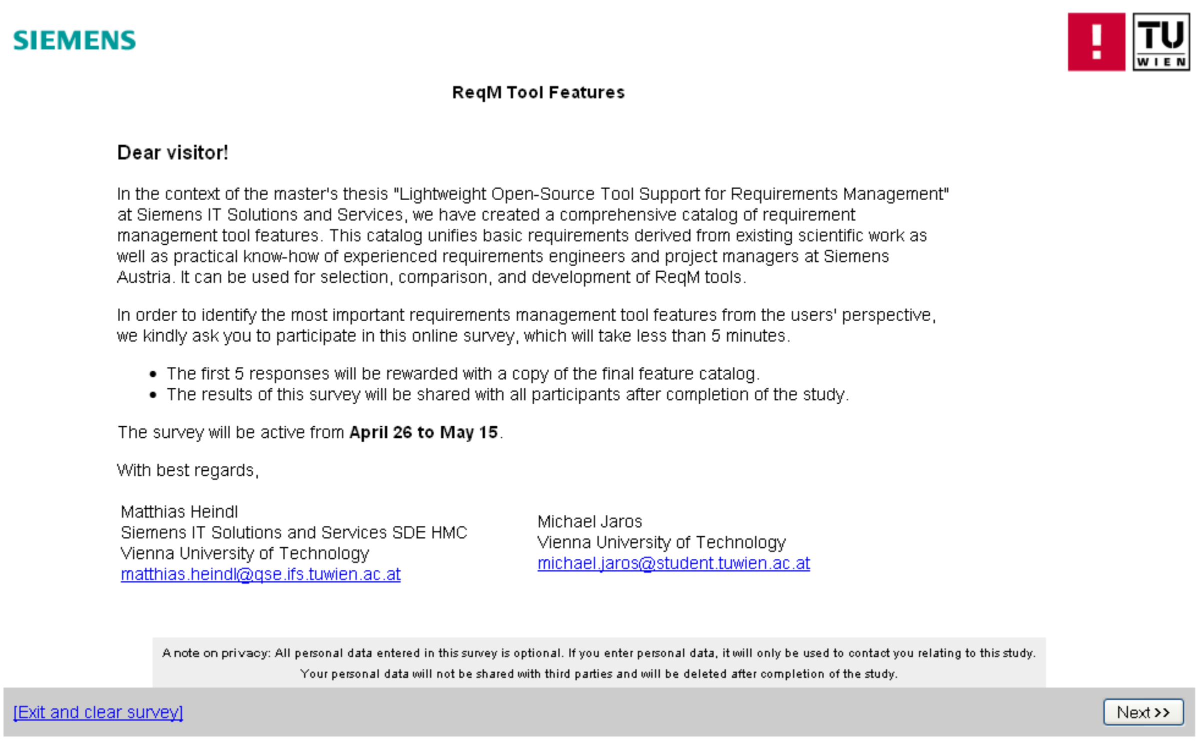
   The organizer selects $n_r=100$ recipients from the ReqM TechnoWeb.

4.1.2.2 Execution of Evaluation

2. The organizer submits the e-mail to all recipients.
3. A subset of the recipients (=participants) go to the web survey and fill out the form.
4. Each participant returns the form via e-mail. The organizer collects all the responses and calculates the **response rate** $r = n_p / n_r$ as the number of responses in relation to the number of recipients.
5. The organizer aggregates all the responses into a single sorted list of features.
6. The organizer classifies the features using a similar granularity as used in the feature catalog. The classification yields a list of feature classes, the number of feature classes s, and the absolute frequencies $f_1 \dots f_s$ for each feature class.
7. The organizer assigns each of the feature classes either to the category “covered” (if it is in the catalog) or to the category “non-covered” (if it is not in the catalog). The organizer calculates the **feature coverage** $u = t / s$ as the number of covered feature classes t in relation to the total number of feature classes s.

4.1.3 Realization and Practical Aspects of the Evaluation

The survey is realized as a web survey that is created using the open-source tool *Limesurvey* and hosted on a dedicated webserver on the internet. The survey is advertised among 453 experts in the fields Requirements Engineering, Test, Project Management and Software Architecture inside the Siemens corporation. The experts are invited via private message in Siemens' technology networking tool. After an introductory screen (Figure 16), the survey collects the participant's field of work and the optional fields name, e-mail address and company. On the third screen, there is only one question asking the participant to provide an ordered list of the ReqM tool features the participant considers most important (Figure 17). There is a large textbox for additional remarks. Filling out the survey should take only about 5 minutes.



SIEMENS

ReqM Tool Features

Dear visitor!

In the context of the master's thesis "Lightweight Open-Source Tool Support for Requirements Management" at Siemens IT Solutions and Services, we have created a comprehensive catalog of requirement management tool features. This catalog unifies basic requirements derived from existing scientific work as well as practical know-how of experienced requirements engineers and project managers at Siemens Austria. It can be used for selection, comparison, and development of ReqM tools.

In order to identify the most important requirements management tool features from the users' perspective, we kindly ask you to participate in this online survey, which will take less than 5 minutes.

- The first 5 responses will be rewarded with a copy of the final feature catalog.
- The results of this survey will be shared with all participants after completion of the study.

The survey will be active from **April 26 to May 15**.

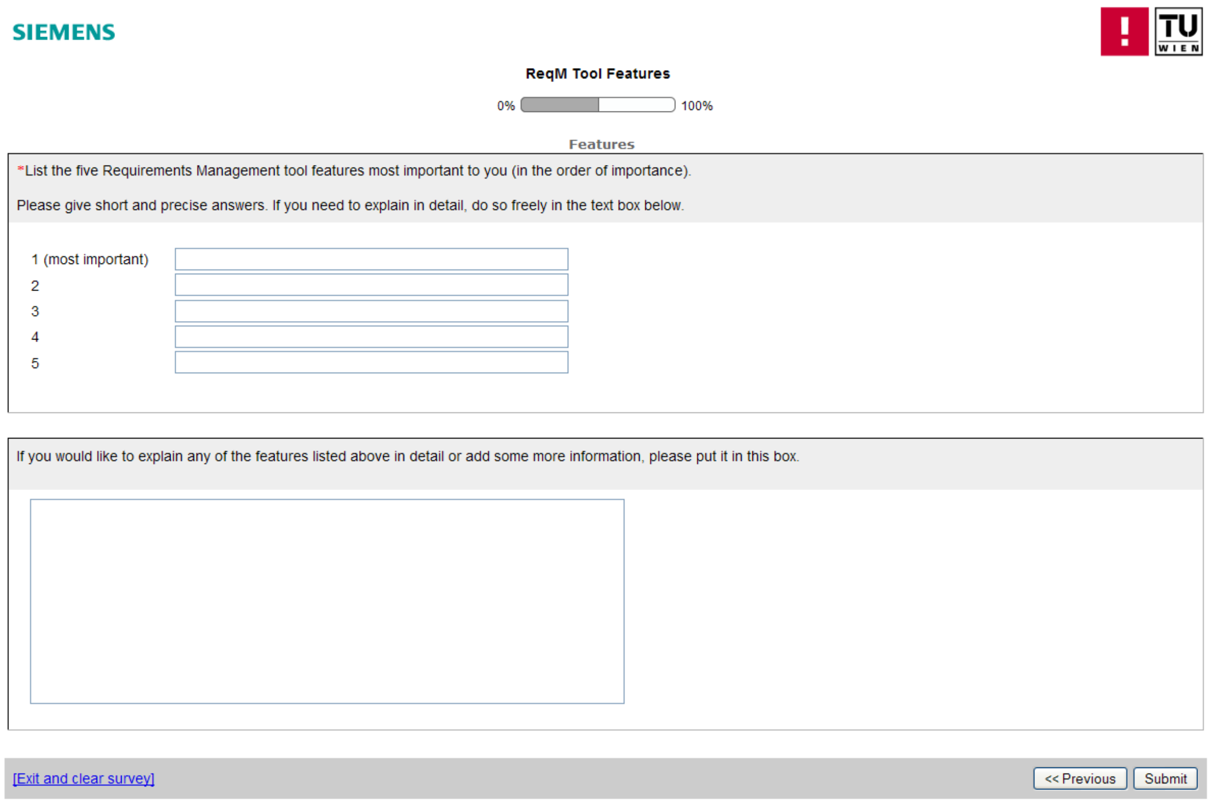
With best regards,

Matthias Heindl
Siemens IT Solutions and Services SDE HMC
Vienna University of Technology
matthias.heindl@qse.ifs.tuwien.ac.at

Michael Jaros
Vienna University of Technology
michael.jaros@student.tuwien.ac.at

A note on privacy: All personal data entered in this survey is optional. If you enter personal data, it will only be used to contact you relating to this study. Your personal data will not be shared with third parties and will be deleted after completion of the study.

Figure 16: Introductory screen of the web survey


SIEMENS

ReqM Tool Features

0% 100%

Features

*List the five Requirements Management tool features most important to you (in the order of importance).

Please give short and precise answers. If you need to explain in detail, do so freely in the text box below.

1 (most important)
2
3
4
5

If you would like to explain any of the features listed above in detail or add some more information, please put it in this box.

[Exit and clear survey]

Figure 17: Only one question (with room for remarks) is posed in the survey.

4.2 Evaluation of the TreqPro Prototype

The TreqPro prototype demonstrates how a lightweight open-source solution can cover essential ReqM features. This part of the evaluation examines, how well the concept implemented in the prototype addresses the identified main shortcomings of existing solutions:

- Limited requirements traceability (see section 1.2.5.1)
- Limited requirements versioning (see section 1.2.5.2)
- Limited ReqM tool integration (see section 1.2.5.3)

4.2.1 Factors Affecting this Part of the Evaluation

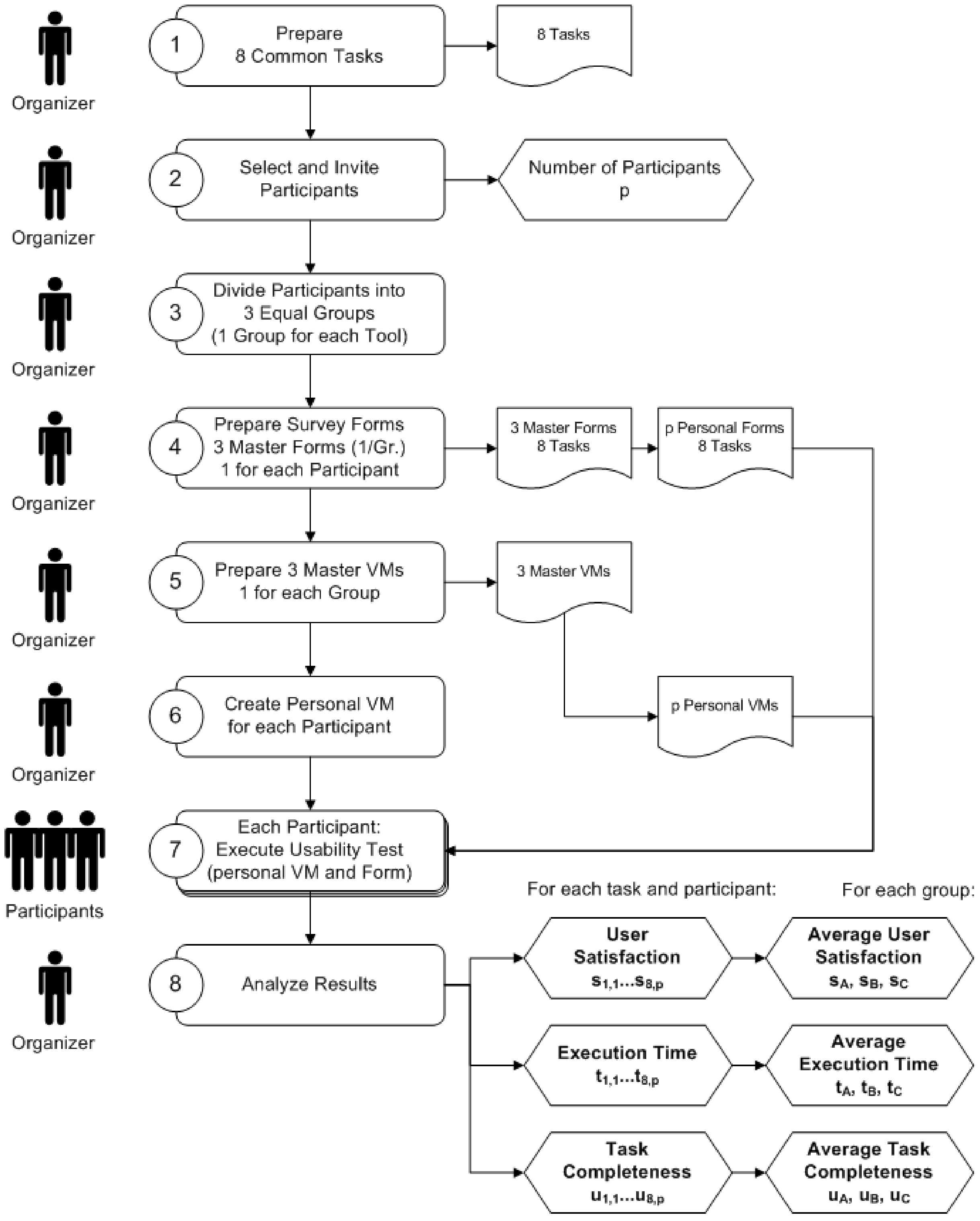
The following factors affect this part of the evaluation (Table 14).

	9 Tasks: The tasks are based on the essential use cases (EUs, see section 3.4.2) and should be carried out by each participant using solution A, B or C, depending on his/her group. Tasks that cannot be carried out due to technical limitations of a solution are removed from all survey forms for that solution.
p	**Number of participants:** Participants will be divided in three equally trained groups, one for each tested solution.
$\mathbf{s_{1,1} \ldots s_{8,p}}$	**User Satisfaction:** The user satisfaction concerning a specific task. This parameter exists for each task and for each participant.
$\mathbf{t_{1,1} \ldots t_{8,p}}$	**Execution Time:** The time necessary to complete a task. This parameter exists for each task and for each participant.
$\mathbf{u_{1,1} \ldots u_{8,p}}$	**Task Completeness:** The completeness of the task after execution (a number from 0 to 1). This parameter exists for each task and for each participant.
Statistical Data: The following measures can be derived from the three basic measures above. They are useful for the actual purpose evaluation because they allow to compare the three groups:	
s_A, s_B, s_C	Average User Satisfaction per Group
$s_{1,A}$, $s_{1,B}$, $s_{1,C}$	Average User Satisfaction per Task and per Group
t_A, t_B, t_C	Average Execution Time per Group
$t_{1,A}$, $t_{1,B}$, $t_{1,C}$	Average Execution Time per Task and per Group
u_A, u_B, u_C	Average Task Completeness per Group
$u_{1,A}$, $u_{1,B}$, $u_{1,C}$	Average Task Completeness per Task and per Group

Table 14: Factors affecting the evaluation of the prototype

4.2.2 Evaluation Method for the Prototype

This section outlines the general process of this part of the evaluation in detail (see Figure 18). For specific details of the realization see section 4.2.3.



Organizer
1 Prepare 8 Common Tasks
8 Tasks
2 Select and Invite Participants
Number of Participants p
3 Divide Participants into 3 Equal Groups (1 Group for each Tool)
4 Prepare Survey Forms 3 Master Forms (1/Gr.) 1 for each Participant
3 Master Forms 8 Tasks
p Personal Forms 8 Tasks
5 Prepare 3 Master VMs 1 for each Group
3 Master VMs
6 Create Personal VM for each Participant
p Personal VMs
Participants
7 Each Participant: Execute Usability Test (personal VM and Form)
For each task and participant:
For each group:
8 Analyze Results
User Satisfaction $s_{1,1}...s_{8,p}$
Average User Satisfaction s_A, s_B, s_C
Execution Time $t_{1,1}...t_{8,p}$
Average Execution Time t_A, t_B, t_C
Task Completeness $u_{1,1}...u_{8,p}$
Average Task Completeness u_A, u_B, u_C

Figure 18: The organizer prepares a personal VM and a personal survey form for each participant. The participants are divided into 3 groups, and each group executes the same tasks with one of 3 tools.

4.2.2.1 Preparation of Evaluation

1. The organizer creates 9 tasks that should be executed with all three solutions, if possible.
2. The organizer selects and invites ReqM experts for the test. *p* is the number of participants.
3. The organizer divides the participants into three groups with a similar RE knowledge / experience level. Each group will test one of the three solutions. The organizer documents the process of finding an optimal group distribution.
4. The organizer prepares 3 master survey forms (one for each group). Each survey form contains:
   - Participant Data
   - General Test Introduction
   - Scenario Description
   - Domain Glossary
   - Task Description for the 9 tasks with
     - Time Measurement and
     - Feedback Form

     Tasks that are not applicable to a solution for technical reasons are grayed out in the survey form for that group.
   - Notes taken during the test by the organizer (appended after the test).
5. The organizer creates 3 master VMs and provides each of them with its own set of test data (one for each group, see Figure 19):
   - Group A: Requisite Pro, Requisite Pro Project and SVN Repository
   - Group B: Trac Environment and SVN Repository
   - Group C: PostgreSQL Database, Trac Environment with TreqPro Plugin and SVN Repository
6. The organizer clones each master VM so that there is one personal VM for each participant. This VM is used for the usability test and will be kept after the test to check the completeness of tasks, if necessary.

4.2.2.2 Execution of Evaluation

7. Each participant executes the usability test:
   a. The organizer hands the participant his/her personal survey form prepared in step 4.

   b. The organizer starts the participant’s personal VM prepared in step 6.
   c. The participant fills out his/her personal data, reads the instructions and scenario information, and carries out each task, recording start and stop time and answering questions in the survey form. After that, the participant fills in the after-test survey.
   d. The organizer takes notes, answers questions and gives hints if necessary.
8. The organizer calculates the results.
   a. The three base factors are calculated for each participant and for each task:
      - User Satisfaction
      - Execution Time
      - Task Completeness
   b. The following extended factors are derived from the base factors:
      - Average User Satisfaction, Execution Time and Task Completeness for each task
      - Average User Satisfaction, Execution Time and Task Completeness for each group

4.2.3 Realization and Practical Aspects of the Evaluation

In concrast to the general, formal description in the previous section, this subsection describes practical details of each evaluation step. The numbers in parentheses indicate the formal step in the previous section that each paragraph refers to, see Figure 18.

4.2.3.1 Preparation of Tasks (1)

From the essential use cases (see section 3.4.2), I have derived 9 tasks for the usability test. Each participant receives a questionnaire containing all tasks.

In this questionnaire, each task is followed by questions for the quantitative parameters satisfaction and time as well as the general qualitative question for free comments. Some of the tasks contain multiple questions for satisfaction for different parts of the tasks so more detailed information about possible shortcomings of the tools can be obtained. The third quantitative parameter (completeness) is obtained separately from the questionnaire.

I will describe each task briefly in the following paragraphs.

Task 1 – Basic Navigation and Management

The participant executes basic ReqM operations such as finding, creating, editing, copying, and moving artifacts.

Task 2 – Forward Traceability

The participant creates a new task ticket, assigns it to a developer, links it to a requirement and finds out about source code associated with another task ticket.

Task 3 – Traceability and Tool Integration

The participant uses a subversion client to commit a source code changeset to the version control repository and reflects the changes in the associated task ticket. The participant also finds out about requirements associated with the task.

Task 4 – Versioning

The participant executes multiple advanced versioning operations including comparison of the working version with an existing baseline, comparison of artifact versions, reverting artifacts to older versions, and branching artifacts.

Task 5 – Simple Query

The participant executes a simple search for an artifact by a given keyword.

Task 6 – Project Template Configuration

The participant modifies the project template configuration by adding an attribute type to an artifact type.

Task 7 – Requirements Document

The participant creates a requirements specification document in the PDF format containing all requirements in the project. This task is followed by additional qualitative questions about applicability of the requirements specification document as well as necessary changes for sharing with management and/or customers.

Task 8 – Semantic Tracing

The participant first creates a new relation type and then uses this relation type to link two artifacts.

Task 9 – Graphical Navigation Interface

In this task, participants navigate from one requirement to another across multiple relations graphically.

4.2.3.2 Selection and Distribution of Participants (2, 3)

The participants are distributed to the three groups in a way as equally as possible mainly concerning tool experience, but also concerning RE experience. None of the participants had any real experience with the prototype tool, but one of the participants in the TreqPro group already knew the prototype a little. Table 15 lists all participants and their experience levels.

Number	Role	RE Experience 0 – 3 (most)	Experience with respective Tool 0 – 3 (most)	Group
1	PM, Design	1,0	0,0	TreqPro
2	RE	3,0	0,0	TreqPro
3	Consulting	2,0	0,0	TreqPro
4	Development	1,0	3,0	Trac
5	PM, Design, Development	1,5	2,0	Trac
6	CM	0,0	1,0	Trac
7	CM	0,5	0,5	Requisite Pro
8	RE	3,0	3,0	Requisite Pro

Table 15: Usability test participant overview

4.2.3.3 Preparation of Survey Forms (4)

The survey forms are created to elicit both quantitative and qualitative input during the usability tests. The forms contain both the tasks (see section 4.2.3.1) and room for answers. They are filled out by each participant before, during and after the usability test.

In addition to the tasks and related questions, the survey form contains the following sections:

Introduction

The introduction describes the basics of the usability test, what to pay attention to and how long it will take.

Test Scenario

The scenario section gives an overview of the fictional project "TechnoWeb 2" in detail so the participant can get an idea of the project.

Glossary

The glossary explains the most important ReqM terms.

Before-Test Survey

- Number
- Group
- Contact Information
- Main Occupation in Software Projects
- ReqM Experience
- Experience with the ReqM Tool Tested
- Expectations towards the Tool

After-Test Survey

- Fulfillment of expectations
- Positive Impressions
- Negative Impressions
- Comments

4.2.3.4 Preparation of VMs (5, 6)

To ensure equal conditions for all the usability tests and to increase replicability, all the tests are executed in virtual machines. The virtual machines are created and executed using Sun VirtualBox 3.1.2r56127 and the following virtual hardware configuration:

- 1 processor
- 512 MB RAM

- 10 GB dynamic disk image
- 20 MB video memory, no hardware acceleration
- PCnet-FAST III (NAT)

VirtualBox does not support cloning VMs yet, so the virtual machine settings are cloned manually using the GUI. Disk images are cloned using the command line tool VBoxManage as follows (but with the absolute paths):

```
VBoxManage clonevdi "source.vdi" "target.vdi"
```

The disk images are handled as follows. First of all a master image with the operating system and the following software is installed:

- MS Windows XP SP3 (German) + all available security updates
- VBox Guest Additions 3.1.2
- Mozilla Firefox (3.5.9)
- MS Office Professional 2003 (German)
- Adobe Reader 9.3 (German)
- Python 2.5.4
    - trac 0.11
    - svn-python 1.6.6
    - setuptools 0.6c9
    - pycopg2-2.0.10
    - genshi 0.5.1
    - eGenix mx base 3.1.2
- PostgreSQL 8.3
- Subversion 1.6.6 (r40053)
- TortoiseSVN 1.6.6.17493

All unneeded GUI elements are disabled. Automatic update reminders and similar notifications are disabled.

After that, the master image is cloned 3 times to create 3 images (one for each tool / group). Each image is configured to match the requirements of its tool. Depending on the group the following additional software is installed:

- TreqPro prototype r945
- IBM Rational Requisite Pro 7.1.1.0

In a third step, each of the three images is cloned so that there is a personal VM image for each participant. This way it is guaranteed that all participants inside a group find equal conditions.

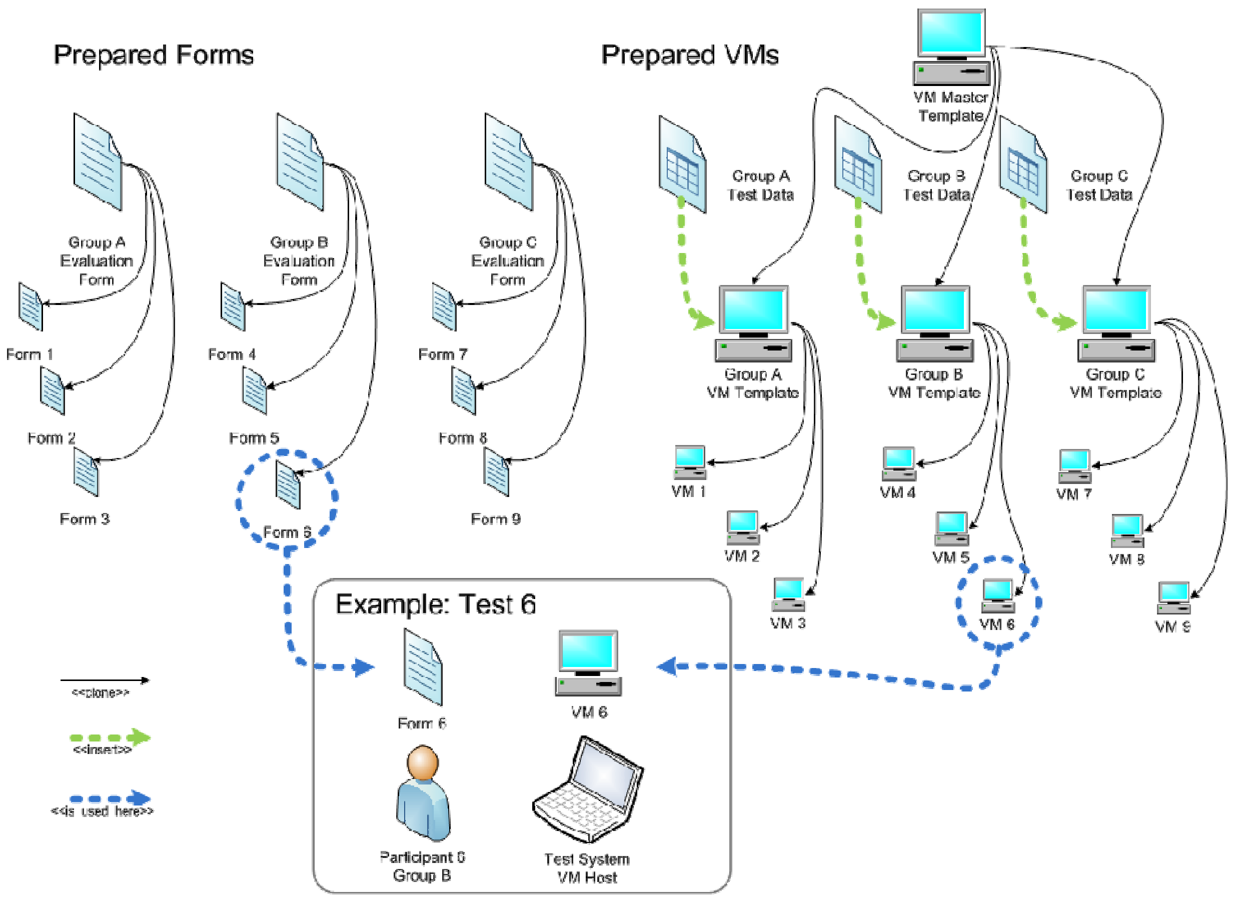


Figure 19: Overview of artifact generation for the usability test

4.2.3.5 Test Execution and Analysis of Results (7, 8)

After completion of the preparations, an appointment is made with each of the participants. I meet the participant in a quiet location, explain the procedure of the test, hand the participant the survey form, start the participant’s personal VM, have him fill in his personal data, and have him execute each task in the VM (recording start and stop time). While the participant executes the steps in each task, I take notes of his actions and comments and answer his questions. After completion of the test, I pose some more questions if anything is unclear and thank the participant for his contribution. I shut down the VM and back it up in a safe place. I permanently attach my notes to the survey form and store them in a safe place.

I collect the quantitative data obtained in each of the 8 usability tests in a spreadsheet document and the qualitative data in a text document for further processing. I calculate statistical measures for the quantitative data, see section 5.2.

4.3 Threats to Validity

I have identified the following threats to validity for each part of the evaluation:

4.3.1 Threats Regarding the Evaluation of the Catalog

- A low response rate (see Table 13 on page 45) increases the chance that the lack of important features in the catalog remains unnoticed. The ideal (but unrealistic) response rate is 1.
- A low feature coverage (see Table 13 on page 45) indicates that important shortcomings of ReqM Tools may have remained unidentified so far. The ideal feature coverage is 1.
- Granularity of the Feedback: Participant feedback at different granularities (e. g. “Create an MS-Word Document” vs. “Create a Requirements Document”) can distort the feature coverage.
- Completeness: Less important features will never be listed because each participant makes a list of only the 5 most important ones.

4.3.2 Threats Regarding the Evaluation of the Prototype

- Generalizability: The results for the small group of participants taking part in the usability test are not necessarily representative for all ReqM users.
- Generalizability: Solutions A and B are the two most used solutions in the Siemens Austria context, however they are only two of many ReqM solutions, and the results might vary in a different context.

- Participant Selection: The participants selected are the ones that are most easily available at Siemens Austria and not necessarily representative for all requirements managers.
- Group Selection: It is difficult to build 3 equally qualified groups. If the qualifications are not balanced well, the results may be distorted.
- Comparability: The common task cannot be executed in each solution in exactly the same way, so approximation has to take place (different features have to be used for the same functionality). The approximation could distort the results, for example if solution X explicitly supports a feature but solution Y can only be used with a workaround for the same task).
- Granularity of the Feedback: It is important that this part of the evaluation reflects the quality of the concept, not the quality of the implementation. Participants will give feedback both on details of the implementation and on the concept which must be treated seperately.
- Participant Expectancies: Test participants might give answers that favor one of the solutions consciously or unconsciously.

5 Results

This chapter describes the results of the evaluation of the tool feature catalog (5.1) and the results of the evaluation of the ReqM tool prototype (5.2).

5.1 Evaluation Results of the Tool Feature Catalog

This section describes the evaluation results of the tool feature catalog. The catalog was evaluated using a web survey. After a summary given in section 5.1.1, the survey response rate is outlined in section 5.1.2. The features are classified in section 5.1.3, the feature coverage is described in section 5.1.4, and features not in the catalog are listed in section 5.1.5.

5.1.1 Summary

I have collected 70 features from the participants of the web survey. 97 % of the features were already covered by the feature catalog. However, due to the low absolute number of participants (14), the results of the web survey cannot be considered representative among all ReqM tool users.

5.1.2 Survey Response Rate

Number of Recipients	453 (RE 148, Test 53, PM 85, Design 167)
Number of Participants	14
Response Rate	3 %

Table 16: Recipients, participants and response rate

5.1.3 Response Feature Classification

From the raw results of the survey, I have created a sorted list which I have then subjected to classification. The *classes* are taken from the main sections of the feature catalog. I have divided these classes into more fine-grained *categories* to get a more exact classification.

5.1.3.1 List of Feature Classes, Categories and Absolute Frequencies

Table 17 lists the classes from the feature catalog in bold letters as well as the corresponding fine-grained categories and the absolute frequencies for each class and category.

Class / Category	Abs. Freq.	
Setup, Customization, Administration and Usability	**29**	
System Properties		9
UI		3
Customization		10
Ease of Use		7
Capturing, Editing and Managing Requirements	**7**	
Collaboration		3
Change Management		4
Configuration Management Aspects	**3**	
Traceability of Requirements	**10**	
Document and Report Generation	**14**	
Import/Export		7
Search, Filtering, Sorting and Reporting		7
Interfaces to Other Tools	**7**	
Costs	**0**	

Table 17: Classification and categories of tool features listed by participants

Both classification schemes are illustrated in Figure 20 and Figure 21.

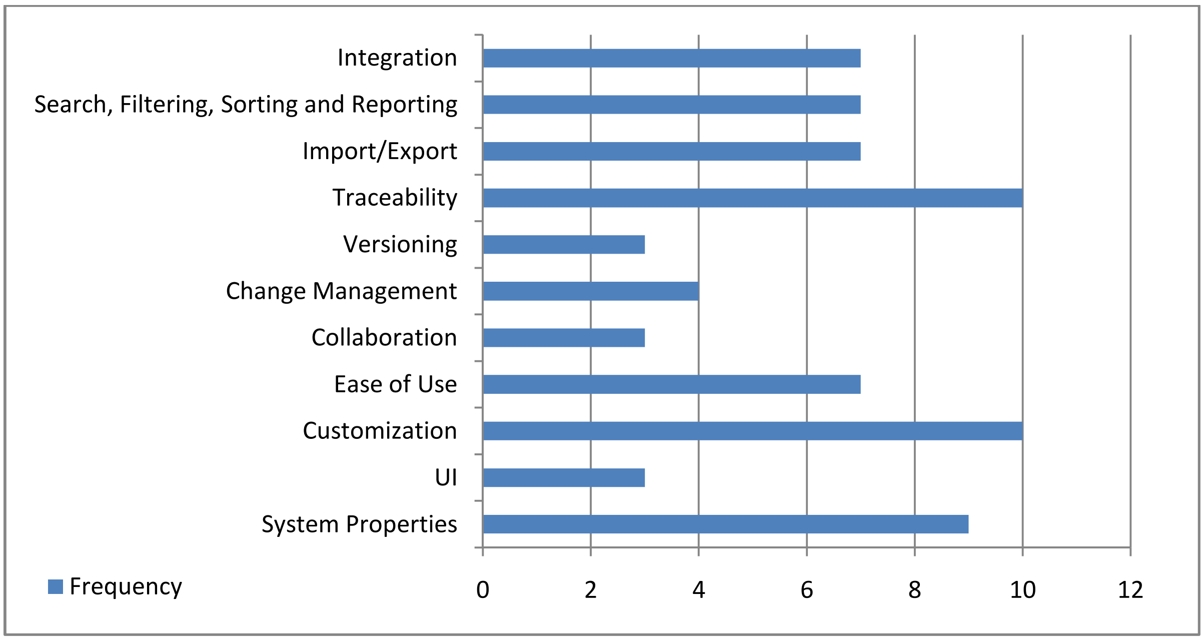


Figure 20: Classification of tool features by new fine-grained categories

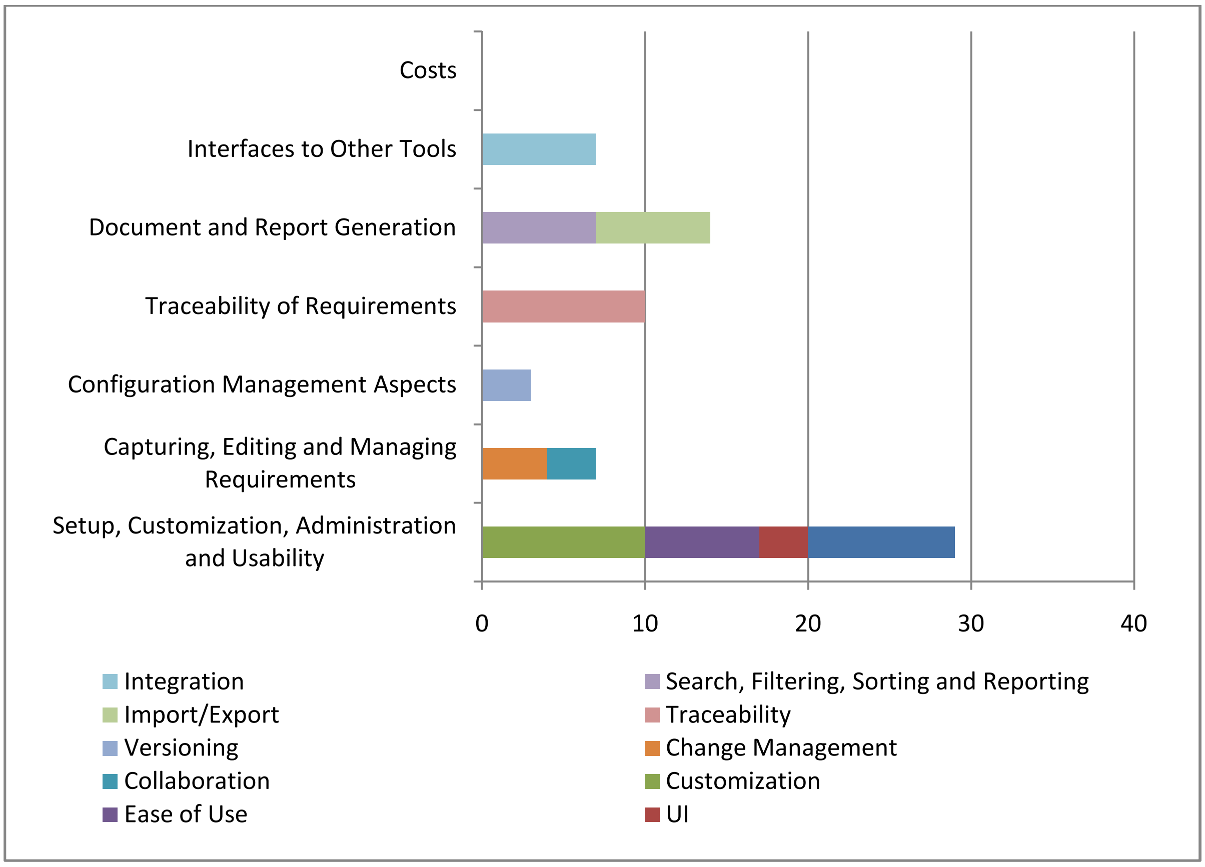


Figure 21: Classification of tool features by coarse-grained classes from feature catalog (x-axis) and new fine-grained categories (colors)

An interpretation of this data is given in the following section.

5.1.3.2 Interpretation of Classification

The features are distributed across the fine-grained categories (Figure 20) in a steady manner. The distribution is not as uniform when considering the more coarse-grained sections of the catalog (Figure 21). This indicates that the first section of the catalog should be divided into several sections.

The frequencies in the fine-grained categories displayed two deviations from the expected pattern:

- 3 of 70 listed features (4 %) belonged to the 'versioning' category. As versioning was a central topic in several expert interviews, I would have expected a higher priorization of this feature class.
- 7 of 70 listed features (10 %) belonged into the 'usability' category and 3 of 70 (4 %) listed features belonged into the 'UI' category. In other words, 14 % of the features were actually non-functional, usability-related requirements. The high priority given to usability-related requirements suggests that the experts consider usability too low in some existing tools.

5.1.4 Feature Coverage

Number of features listed by participants and covered in catalog	68
Total number of listed features	70
Feature Coverage	97 %

Table 18: Feature Coverage

5.1.5 Features not in the Catalog

There were 2 specific features that were not included in the catalog:

- Refactoring of requirements
- SAP integration (e. g. for indicating requirement costs to the sales department)

5.2 Evaluation of the TreqPro Prototype

The evaluation of the prototype was carried out as a usability test as described in section 4.2. The usability tests yielded both qualitative and quantitative results for each of its 9 tasks. This section first gives an overview of the results (section 5.2.1) and then presents the detailed results for each task in the sections 5.2.2 – 5.2.10.

The following quantities were recorded in this evaluation:

satisfaction (s) – A measure of usability of a specific application feature as experienced by a participant. The satisfaction is given on a scale from 0 (worst) to 3 (best) rounded to 2 decimal places. 3 (very easy) means as much as it could not be any easier to use, and 0 (very difficult) means that it could not be any more difficult or sophisticated to use (within reasonable boundaries).

time (t) – Time it took the user to complete a task. The time was measured by the participant himself and checked by the test manager using the clock in the test VM's task bar. Special care had to be taken to record only working time and stop the watch if the participant started talking about anything not related specifically to the current task. The time is given in minutes rounded to 30 seconds.

completeness (c) – Percentage of the task completed by the user. Each task consisted of several subtasks. Each subtask could be completed not at all (0), half (0.5) or full (1). The completeness for a task was calculated as the arithmetic mean of all its subtasks. The completeness is given in percent rounded to 0 decimal places.

arithmetic mean (AM) – The arithmetic mean has the same unit and precision as the respective quantity.

coefficient of variation (CV) – The coefficient of variation is the average deviation from the arithmetic mean in percent of the arithmetic mean (rounded to 0 decimal places) and allows to compare deviations on the different scales of the 3 parameters s, t and c.

5.2.1 Summary of Results

This section summarizes all quantitative (section 5.2.1.1) and qualitative (section 5.2.1.2) results of the evaluation of the prototype.

5.2.1.1 Average Quantitative Results

Only tasks 1 to 5 could be completed (i. e. $c > 50$ %) with all 3 tools. Therefore only these tasks were used to compare satisfaction and time, but all tasks were used to compare completeness.

1. The arithmetic mean (satisfaction and completeness) or the sum (time) was calculated for each participant over all compared tasks.
2. The arithmetic mean and coefficient of variation were calculated over all participants in that group. The CVs indicate the deviation from the AM inside the respective groups.

Table 19 gives an overview of the quantitative results:

Task	TreqPro AM	TreqPro CV	Trac AM	Trac CV	Requ. Pro AM	Requ. Pro CV
Ø $s_{1\text{-}5}$	2,43	12 %	1,74	28 %	1,51	15 %
$\sum t_{1\text{-}5}$	31 min	21 %	47 min	21 %	51 min	11 %
Ø c	99 %	2 %	53 %	9 %	57 %	5 %

Table 19: Summary of quantitative results

The following subsections explain the summarized results for each of the 3 quantities satisfaction, time and completeness. The detailed results for each task can be found in sections 5.2.2 – 5.2.10.

Satisfaction

This subsection gives an overview of the average satisfaction results for all the tasks.

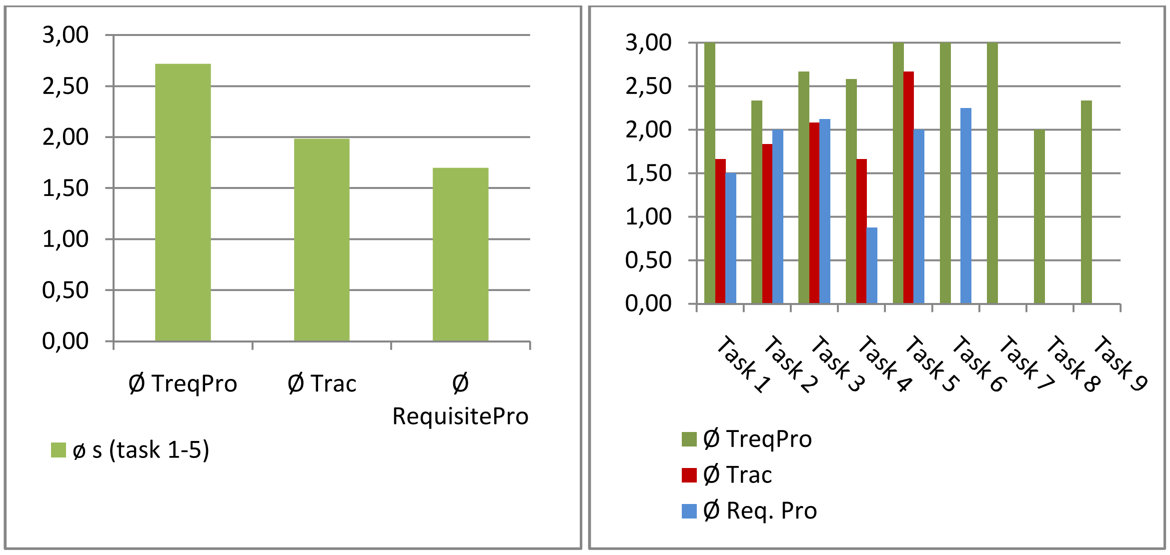


Figure 22: Average satisfaction (s) for each tool and task

The results clearly show a significantly higher satisfaction of participants with TreqPro compared to the other tools. The open source tool Trac showed about the same satisfaction as the commercial tool Requisite Pro.

The coefficient of variation for the satisfaction was relatively high inside the Trac group, which I investigated further: It turned out that the two users with less Trac tool experience were much more satisfied (2.55 and 2.05) with the results than the Trac expert (1.35), even though they could complete less of the given tasks (49 % and 53 %) compared to the expert (58 %).

The highest advantage (i. e. difference to the closest other tool) was measured in basic requirements operations with a difference of 1.33 (on a scale from 0 to 3), followed by versioning with a difference of 0.92 and project template configuration with 0.75.

With the prototype, satisfaction could be improved by 40 % compared to the existing tool Trac and by 61 % compared to the existing tool Requisite Pro.

Time

This subsection gives an overview of the average time results for all the tasks.

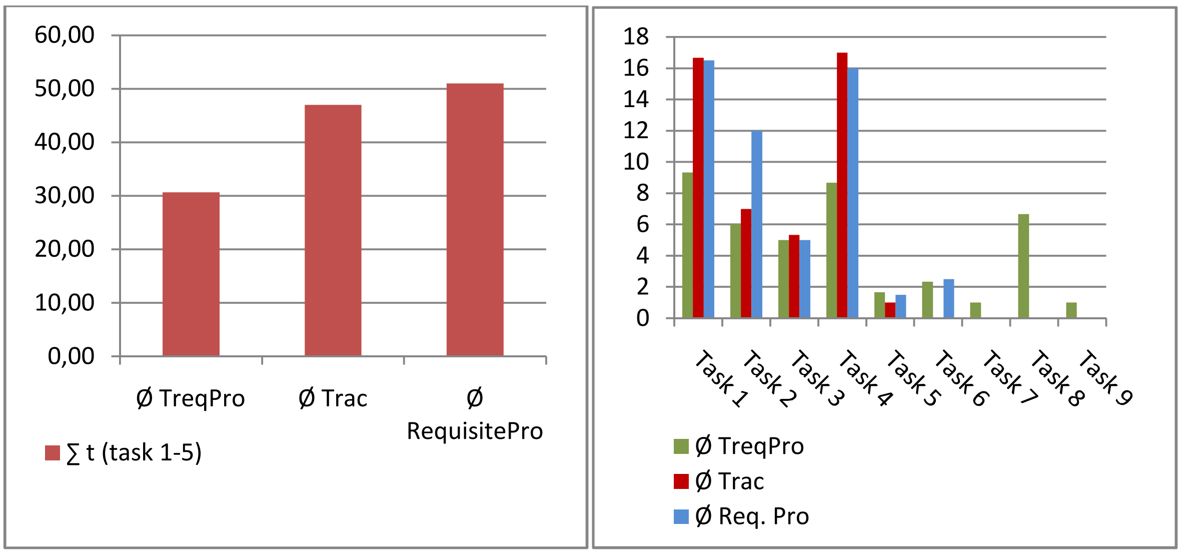


Figure 23: Average execution time (t) for each tool and task

The results show that TreqPro performed significantly (40 – 50 %) better than at least one of the other tools in tasks 1, 2 and 4, i. e. it took participants only about half the time to carry out basic requirements operations, establish forward traceability and use different versioning functions. There was no significant difference in execution time in tasks 3, 5, and 6 which consisted of tool integration, search for specific requirements, and project template configuration.

The average total time for all tasks with TreqPro was 35 % smaller than with Trac and 40 % smaller than with Requisite Pro.

With the prototype, execution time could be improved by 34 % compared to the existing tool Trac and by 39 % compared to the existing tool Requisite Pro.

Completeness

This subsection gives an overview of the average completeness results for all the tasks.

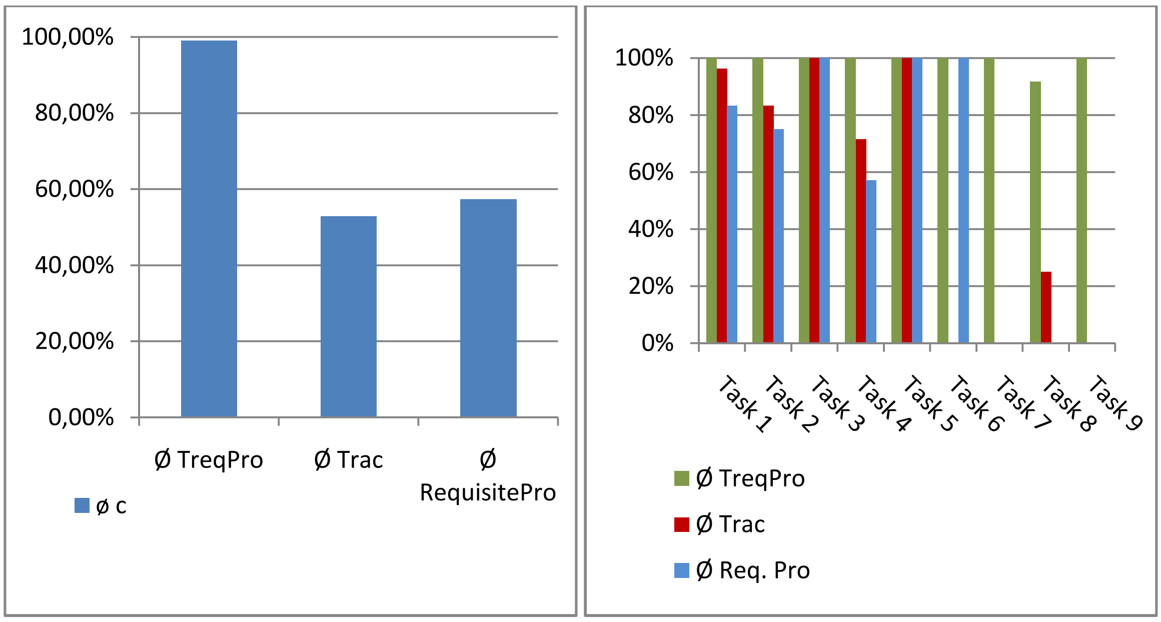


Figure 24: Average completeness (c) for each tool and task.

The average completeness with TreqPro was 99 %, while it was only 53 % with Trac and 57 % with Requisite Pro. Only one participant could not complete one task (Task 8) with TreqPro because of usability issues.

TreqPro was the only tool in which tasks 7, 8 and 9 could be completed (c > 50 %). Task 6 could not be completed in Trac.

With the prototype, completeness could be improved by 87 % compared to the existing tool Trac and by 74 % compared to the existing tool Requisite Pro.

5.2.1.2 Qualitative Feedback

This subsection gives an overview about qualitative results of the usability test that has been aggregated and grouped by the main fields of interest (traceability, versioning, tool integration).

Qualitative Feedback on TreqPro

Traceability

Display and creation of traces does not yet seem to be as intuitive as desired. Minor improvements, such as moving the relevant command from the relations tab to the context menu, will probably address this.

Versioning

TreqPro has the most advanced versioning capabilities and allows extended techniques like baselining and branching with a fair amount of usability. Not all users however require full versioning support. Instead some would rather have more workflow support. The comparison of both artifact and project versions is straightforward.

Tool Integration

The integration with issue tracking and version control worked seamlessly.

User Interface, Basic Operations, and Document Generation

TreqPro has the best user interface of the three tools, even though not all planned features (drag & drop, artifact browser …) have been implemented fully. Basic Copy/Move operations took much more effort in both other tools, mainly because requirements cannot be directly edited without limitations. TreqPro allows generating a requirements specification document in the PDF format containing all requirements. However, the exact contents of the document are not configurable yet. TreqPro contains an experimental implementation of a graphical requirements browser.

Qualitative Feedback on Trac

Traceability

Establishing task-source traceability is a manual, error-prone process in Trac, but it is possible to obtain a high level of traceability due to the integration of version control.

Versioning

Trac’s wiki versioning is linear and does not support baselining or branching, however workarounds for baselining are possible. Users could choose to use tickets instead of wiki pages to store requirements. Trac tickets support workflows and allow for better structuring than wiki pages, but they do not support versioning, which makes baselining very difficult.

Extended techniques such as baselining and branching are only possible through workarounds which are hard to understand for users. While Trac allows for easy comparing of two requirement versions, it is difficult to compare whole project versions.

Tool Integration

The integration with issue tracking and version control worked seamlessly.

User Interface, Basic Operations, and Document Generation

Trac does not offer project templates, which leads to inconsistent structuring and formatting of requirements. On the other hand it is quite a simple tool that gives much freedom to its users. Multiple browser windows or tabs are necessary to do copy/move operations quickly, but these operations are still error-prone. Trac does not offer automatic generation or checking of requirements IDs. Trac is generally easy to use, but as the tests showed, it can be sophisticated for new users. Trac does not come with any document export functionality.

Qualitative Feedback on Requisite Pro

Traceability

It is not possible to maintain m:n relations across tool borders with Requisite Pro in a practical way. Relations inside Requisite Pro can be created, but they cannot have a type.

Versioning

In Requisite Pro it is difficult to identify changes between requirements, and it is even more difficult and inconvenient to compare project versions. In both cases, the differences are not highlighted, so minor changes are difficult to spot.

Tool Integration

Only little integration was possible with the issue tracking and version control systems. Users like the integration with MS Word, which works well in many basic scenarios, but leads to many problems on the other hand (e. g. search and change management do not work fully any more, copy/move operations are more difficult).

User Interface, Basic Operations, Collaboration and Document Generation

Requirements in Requisite Pro can be copied and moved conveniently, if they are stored in the database only. If requirements are stored in word documents, copying and moving is tedious. While the basic concept and database of Requisite Pro are sufficient for most small and medium-sized projects, its user interface is its great weakness. Direct editing of requirements is sometimes difficult because of many small UI problems. Requirements that are not in the database, but in a Word document, are not included in the change history. Collaboration on one project is difficult in Requisite Pro, especially if Word documents are used, because only one user can edit each document at a time. Requisite Pro cannot be easily customized; extension of its functionality is possible through the extensibility interface. The search for specific requirements can be difficult, because only one search result is displayed at a time, and because the database and each document must be searched separately. Requisite Pro allows generation of one report per requirement type in the DOC format. Document generation is inconvenient and has severe bugs, but external tools (like SODA) can be used for better results. The menus are not well-structured; therefore it takes new users some time to find the desired functionality (e. g. project template configuration).

5.2.2 Task 1 – Basic Navigation and Management

5.2.2.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	3,00	0 %	1,67	34 %	1,50	47 %
t	9,33	34 %	16,67	24 %	16,5	21 %
c	100 %	0 %	96 %	3 %	83 %	28 %

Table 20: Quantitative results for Task 1

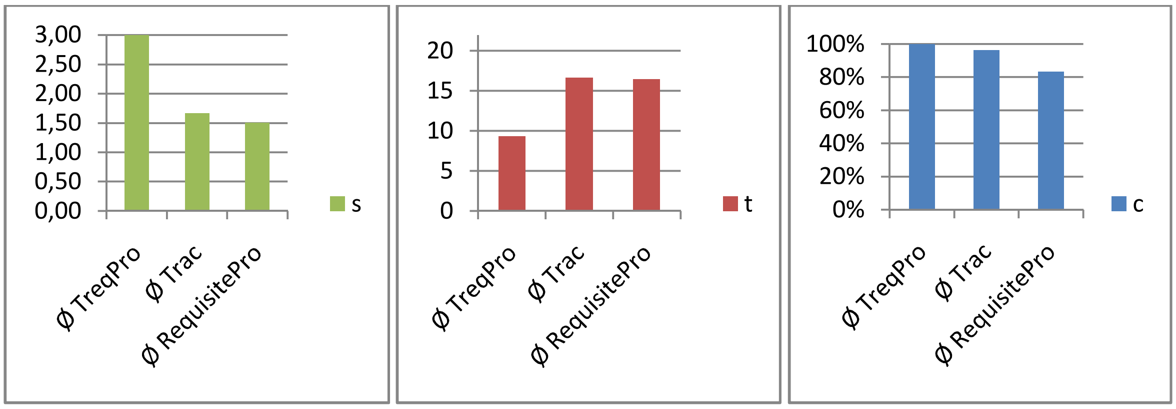


Figure 25: Average satisfaction (s), execution time (t) and completeness (c) for Task 1

5.2.2.2 Qualitative Feedback

TreqPro

- 3 of 3 participants especially mentioned TreqPro's good user interface.
- 2 participants used the query box to find artifacts quickly.
- The lack of drag & drop functionality confused participants at first glance. After they first tried it, all participants described the cut/paste method as intuitive.
- All participants liked UI features like text box resizing for easier requirements editing.
- It was very easy for participants already familiar with Trac to use the wiki syntax for requirements descriptions.
- Possible improvements:
  - less verbose notices
  - automatic restriction and more filtering options on search results

Trac

- All 3 participants stated that copy and move operations take a lot of time because all page names and links must be modified manually multiple times (which is redundant and error-prone).
- All 3 participants used more than one browser window or tab for copy and move operations.
- All 3 participants complained about lack of project template support, which resulted in inconsistently structured requirements.
- 2 participants less familiar with Trac would like to have WYSIWYG support for wiki text editing.
- 2 participants were missing automatic generation and duplicate-checking of requirement IDs.
- 2 participants mentioned the lack of workflows in Trac wiki pages as used in the example. They stated that they would rather use tickets instead of wiki pages for requirements management, even though Trac tickets do not offer versioning and it would not be possible to create baselines of requirements.
- 1 participant stated that he considered Trac suitable even for projects with many requirements, but that a high level of tool experience would be required.

Requisite Pro

- 2 of 2 participants concluded that copying and moving requirements is inconvenient with Requisite Pro.
- 2 participants stated that the tool has bad usability. Some features are missing, some features are patched together and integrated badly (e. g. comparison of baselines).
- 1 participant described MS Word integration as a good and intuitive feature, but this integration has drawbacks: The synchronization between the RequisitePro database and the word documents works one-way only, i. e. if the structure in the database is modified, the documents are not updated. Also, renumbering of requirements does not work automatically (which could be seen as a feature).
- Usability for requirement editing in RequisitePro is very cumbersome:
    - 1 participant mentioned the editing of multi-line attributes in a single-line textbox.

    - 1 participant demonstrated bad mouse wheel support in dialog windows.
    - Formatting of requirements is possible in the Word documents, but not in the database.
- Only requirements in the database are included in the change history. To include more data in the change history, one workaround would be to define one requirement type for each field of a requirement, but this is very inconvenient and makes report generation almost impossible (because a report can only be created for one requirement type at a time).
- 1 participant described that when working together on requirements, it is better to use multiple documents because of file locking.

5.2.3 Task 2 – Forward Traceability

5.2.3.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	2.33	33 %	1.83	32 %	2	35 %
t	6.00	29 %	7.00	38 %	12	35 %
c	100 %	0 %	83 %	17 %	75 %	0 %

Table 21: Quantitative results for Task 2

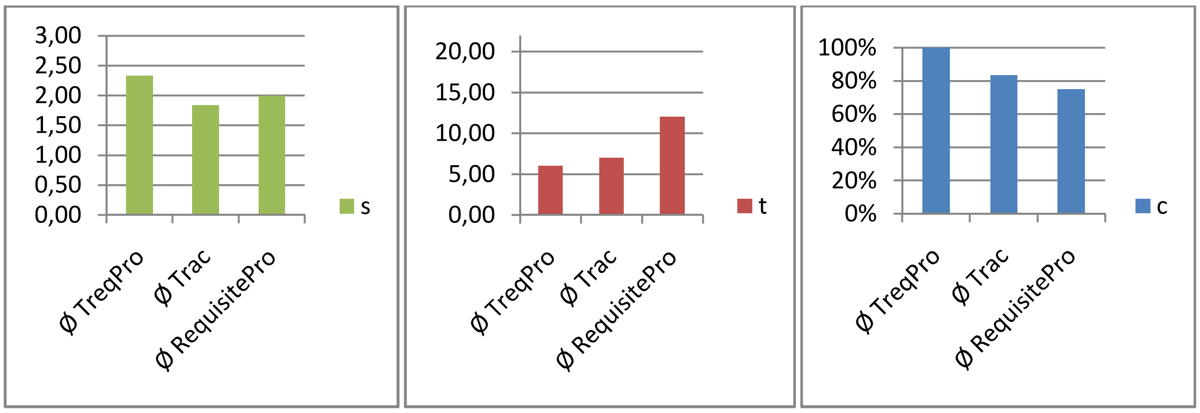


Figure 26: Average satisfaction (s), execution time (t) and completeness (c) for Task 2

5.2.3.2 Qualitative Feedback

TreqPro

- It took 2 of 3 participants some time to find the tickets associated with an artifact.
- 2 participants explicitily mentioned TreqPro's good user interface during this task.
- 2 participants associated ticket and artifact from the ticket actions box, one user did it the other way around using the controls in the artifact's tickets box.
- 1 participant noted that the list for artifact selection in ticket action box does not have any filters and will be to long in large projects (TreqPro's built-in artifact browser provides a solution for this, but it has not been integrated into all parts of the application yet).
- 1 participant suggested to allow creating traces via context menu („trace to").

- 1 participant suggested that direct view of linked source files is better than the indirect way over tasks.

Trac

- All 3 participants used either more than one tab or more than one window in the web browser in this task.
- 2 participants noted that it is very inconvenient to create links with Trac, especially with bidirectional relations, mainly because of the necessary redundant, manual changes.
- 1 participant noted that this task can be accomplished very well with Trac if the user has enough experience with this tool.
- 1 participant created absolute links between artifacts which can lead to problems if the project‘s URL changes.
- 1 participant (who did not have any Trac experience, but technical background) did not manage to link a ticket and a wiki page even with repeated, detailed instructions.

Requisite Pro

- All 2 participants managed to add links from MS Word documents to Trac tickets which resulted in unidirectional "clickable" traceability from the requirements documents to the tickets.
- All 2 participants gave suggestions for traceability from Trac to Requisite Pro, but the user would have had to look up the reference manually in every case.
- 1 of 2 participants described RequisitePro as unsuccessful (literally: "nicht zielführend").
- 1 participant described a workaround for addition of a custom field to trac tickets for the relation (custom fields in ticket sections in trac.ini). The user confirmed the test manager’s objections that this would not work for m:n relations and that there would not be a clickable bi-directional link.
- 1 participant described the option of adding an attribute for the Trac ticket URL, but this would not be possible for more m:n relations (more than 1 URL).

5.2.4 Task 3 – Traceability and Tool Integration

5.2.4.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	2.67	22 %	2.08	30 %	2.13	8 %
t	5.00	0 %	5.33	29 %	5.00	28 %
c	100 %	0 %	100 %	0 %	100 %	0 %

Table 22: Quantitative results for Task 3

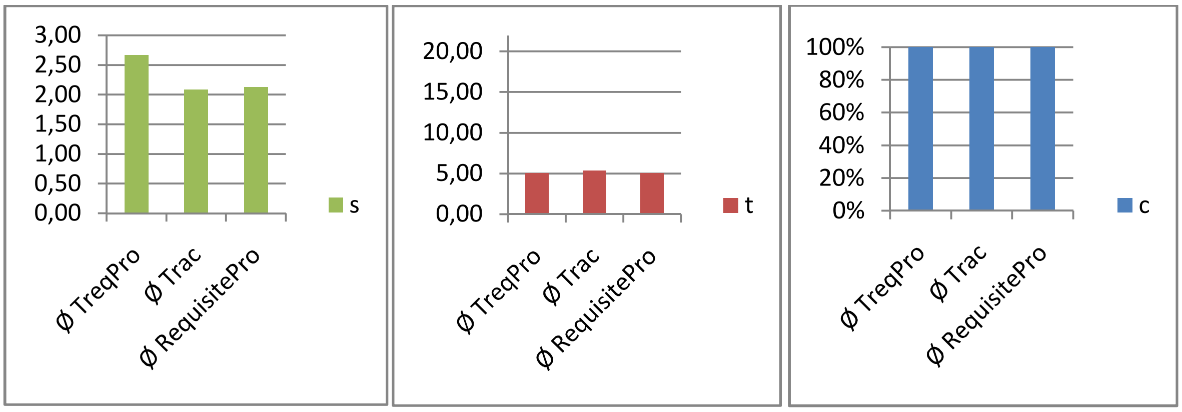


Figure 27: Average satisfaction (s), execution time (t) and completeness (c) for Task 3

5.2.4.2 Qualitative Feedback

TreqPro

- 2 participants suggested displaying the artifact name instead of id and version in hyperlinks.
- 1 participant liked the example project ("extremely nice").
- 1 participant expressed general appreciation of the tool during this task ("super").
- 1 participant noted that a selection of open tickets would be very useful when constructing the commit message.
- 1 participant asked how he could find a ticket most easily. The test manager suggested entering the ticket number in the search box in the top right corner.

Trac

- 1 participant explained that there are many steps necessary to establish task-source traceability (commit, find ticket number, find or store revision number, manually create link in ticket). He added that the method is error-prone in projects with many tickets and/or many commits. He also noted that the linking parts (revision in ticket, ticket in revision) of the traceability process may be accidentally left out because the process is not enforced in the workflow.
- 1 participant did not store or remember the revision number of the commit as requested in the task and used the Trac timeline to look up the revision number manually.

Requisite Pro

- 1 participant stated that a direct relation / trace link cannot be created across tool borders (between Requisite Pro and Trac).
- All 2 participants did not store or remember the revision number of the commit as requested in the task and used the Subversion repository browser to look up the revision number.

5.2.5 Task 4 – Versioning

5.2.5.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	2.58	20 %	1.67	53 %	0.88	20 %
t	8.67	35 %	17.00	26 %	16.00	18 %
c	100 %	0 %	71 %	36 %	57 %	0 %

Table 23: Quantitative results for Task 4

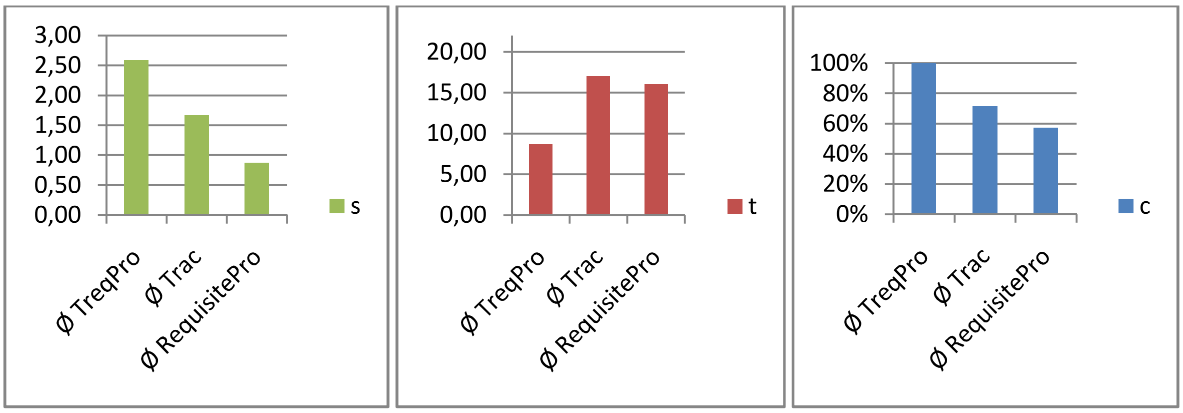


Figure 28: Average satisfaction (s), execution time (t) and completeness (c) for Task 4

5.2.5.2 Qualitative Feedback

TreqPro

- All 3 participants liked the free switching between existing versions of an artifact ("switch back is very easy", "na bumm, das geht?", "das ist aber toll").
- 1 participant stated that the diffs are difficult to read if the way of displaying differences with the color scheme is unknown to the user. The same participant first had some problems identifying the nature of changes (added vs. modified), but then described the diff capabilities of the application as sensational ("spektakulär").
- 1 participant stated that the user interface should give some hints on how to do the branching of artifacts.
- 1 participant would prefer a one-column radio button user interface for the selection of versions to compare.

Trac

- All 3 participants were confused by the implementation of baselines used in the example project (wiki page with links to specific versions of each requirement) even though the concept was explained in detail by the test manager.
- 2 participants concluded that the comparison of project versions (baselines) is very difficult in Trac, as every page has to be checked manually (because the current version of a wiki page is not visible 'from outside' via its link as a specific version is). 1 participant did not manage to compare two project versions.
- 2 participants stated that changes on a single requirement can be tracked very well using Trac's wiki compare functionality.
- 2 participants accidentally compared two wrong versions of a requirement.
- 2 participants stated that restoring of previous versions is easy if the changes can be permanently deleted, but difficult if they should be preserved.
- 2 participants stated that branching is very difficult in Trac, as it can only be accomplished using workarounds and the results cannot be used practicably. 1 participant could not think of any workaround for branching.
- Trac displays the wiki source code of the latest version only, which makes manual "branching" even more difficult (because the rendered text has to be taken and reformatted if an old version is needed).

Requisite Pro

- When looking for baselines, 1 participant checked for milestones in the Trac ticket system (but none were present in the example project).
- 1 participant stated that going back in previous versions is not possible in Requisite Pro.
- 1 participant described Requisite Pro as "far too circumstantial" as far as versioning support is concerned and added that some problems (e. g. going back to old versions, branching) just could not be solved.
- When comparing baselines, all 2 participants first tried to view the last baseline, then stated that they had to create a new baseline to compare the existing one to the current state of the project.

- When creating a new baseline, all 2 participants had minor difficulties and got multiple error messages when configuring the directories for baseline creation because of the sub-optimal user interface for directory selection. 1 participant stated that this functionality is "not too easy to use". All 2 participants got a concurrency error message and had to close the project to create the baseline.
- When comparing baselines, 1 participant stated that a colored view of differences "would be great".
- All 2 participants stated that that TreqPro does not allow going back to old versions of a requirement. 1 user stated that it is possible to go back manually, and that this workaround would be very impractical for many changes and/or many versions.
- 1 participant stated that the comparison of project versions is implemented badly, but acceptable ("zumutbar") for experienced RequisitePro users.

5.2.6 Task 5 – Simple Query

5.2.6.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	3.00	0 %	2.67	22 %	2.00	0 %
t	1.67	69 %	1.00	0 %	1.50	47 %
c	100 %	0 %	100 %	0 %	100 %	0 %

Table 24: Quantitative results for Task 5

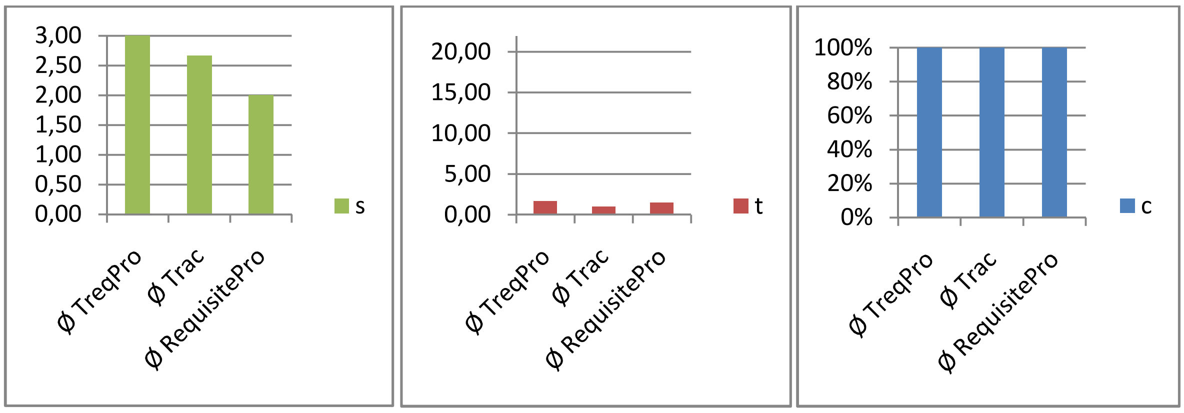


Figure 29: Average satisfaction (s), execution time (t) and completeness (c) for Task 5

5.2.6.2 Qualitative Feedback

TreqPro

- 1 participant stated that better filtering of results (e. g. by artifact types) is necessary.

Trac

- 1 participant stated that the search is easy, but better filtering of results (e. g. by artifact types) is necessary.

RequisitePro

- 2 participants stated that a list of search results would be much better than jumping between the results.
- 1 participant mentioned that filtering by type would be a good feature.

5.2.7 Task 6 – Project Template Configuration

5.2.7.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	3.00	0 %			2.25	47 %
t	2.33	66 %			2.50	85 %
c	100 %	0 %	0 %	0 %	100 %	0 %

Table 25: Quantitative results for Task 6

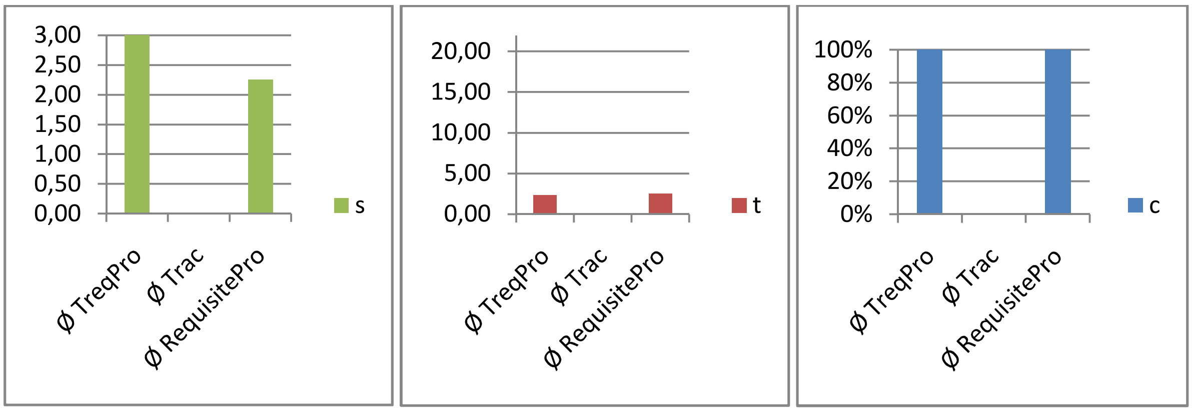


Figure 30: Average satisfaction (s), execution time (t) and completeness (c) for Task 6

5.2.7.2 Qualitative Feedback

TreqPro

- 1 participant stated that he would have expected the project template configuration in Trac's admin area rather than in the TreqPro menu.

Trac

- All 3 participants concluded that Trac does not allow configuring a project template and that therefore the task cannot be performed.
- 1 participant suggested using the Trac plugin PageTemplates to support entering requirements in a predefined form.
- 2 participants suggested to use tickets instead of wiki pages because
  - custom fields can be defined for tickets and

    - tickets have a workflow,

even though Trac tickets do not support versioning. Both participants agreed that tickets cannot be used if versioning or baselining is required.

Requisite Pro

- 1 participant had severe difficulties finding the project template configuration.

5.2.8 Task 7 – Requirements Document

5.2.8.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	3.00	0 %				
t	1.00	100 %				
c	100 %	0 %	0 %	0 %	0 %	0 %

Table 26: Quantitative results for Task 7

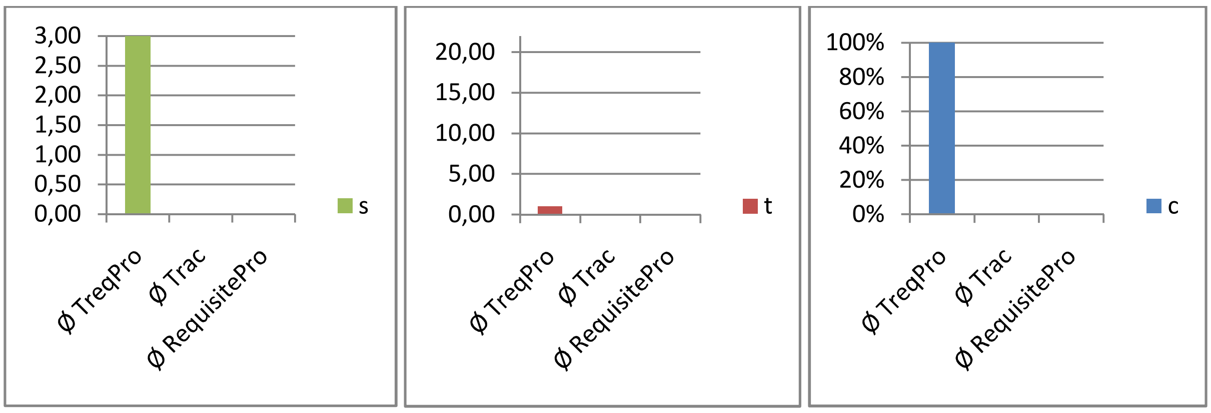


Figure 31: Average satisfaction (s), execution time (t) and completeness (c) for Task 7

5.2.8.2 Qualitative Feedback

TreqPro

- It took 1 participant some time to find the document generator.
- 1 participant liked the quick and easy generation process.
- 1 participant suggested to add comprehensive configuration options:
  - tabular or full-text view
  - visibility and position of each attribute and relation
  - traceability information
  - free content
  - multimedia content
  - templates with placeholders
- The participants stated that the document could be used

    - for negotiation with customers (2 participants)
    - in reviews (2 participants)
    - as an overview (1 participant)
    - as an attachment to a software requirements specification (1 participant)
    - as a specification for designers (1 participant)

Trac

- All 3 participants concluded that automated document generation is not possible with Trac, unless a plug-in is used, e. g. the document generator by Alexander Wagner.
- 2 participants added that plain-text export of single wiki pages is possible with Trac.

Requisite Pro

- 1 participant did not know how to create a document.
- 1 participant stated that it is not easily possible with Requisite Pro, but described the following workaround:
    - For each requirement type:
        - Create a view.
        - Restrict visible fields.
        - Select "File / Export to Word".

  This workaround does not work together with the fields workaround described in task 1. Furthermore there is a bug that creates too many page breaks if there are many columns, so that the resulting document is not usable. The participant added that the resulting document must usually be edited extensively to be of any use.
- 1 participant mentioned the external tool Soda which can be used for document and report generation.

5.2.9 Task 8 – Semantic Tracing

5.2.9.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	2.00	50 %	$^{1)}$	$^{1)}$		
t	6.67	57 %	$^{1)}$	$^{1)}$		
c	92 %	16 %	25 %	0 %	0 %	0 %

$^{1)}$ s and t were not comparable, as only part of this task could be done with this tool.

Table 27: Quantitative results for Task 8

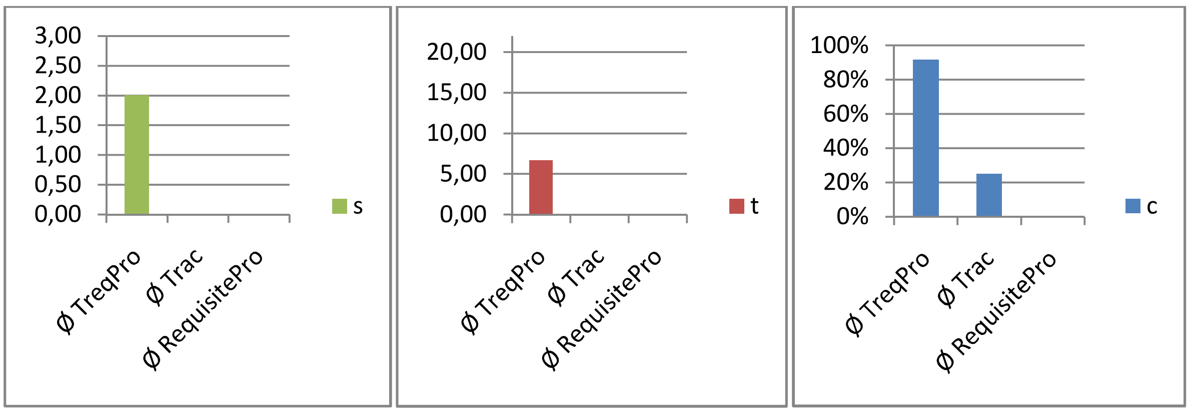


Figure 32: Average satisfaction (s), execution time (t) and completeness (c) for Task 8

5.2.9.2 Qualitative Feedback

TreqPro

- 2 participants were slightly confused by the controls in the artifact browser (filters button does not disable filters when hidden, scrollbar almost hidden, cancel button far at the bottom).
- 1 participant managed to create a relation, but never with the specified type (because he mistook the graphical view relation type selection for the relation type selection for new relations).
- 1 participant used the trace matrix to create the relation, 1 participant used both the trace matrix and the target selection. The participant that tried both methods liked the trace matrix better.

Trac

- All 3 participants could create bidirectional links manually. 1 participant used absolute links that will break if the project's URL changes.
- All 3 participants concluded that Trac does not support semantic traces. 1 participant demonstrated that basic semantics can be added as link text.

Requisite Pro

- All 2 participants concluded that Requisite Pro does not support semantic traces.
- No participant could come up with a workaround.

5.2.10 Task 9 – Graphical Navigation

5.2.10.1 Quantitative Measures

	AM TreqPro	CV TreqPro	AM Trac	CV Trac	AM RequisitePro	CV RequisitePro
s	2.33	25 %				
t	1.00	0 %				
c	100 %	0 %	0 %	0 %	0 %	0 %

Table 28: Quantitative results for Task 9

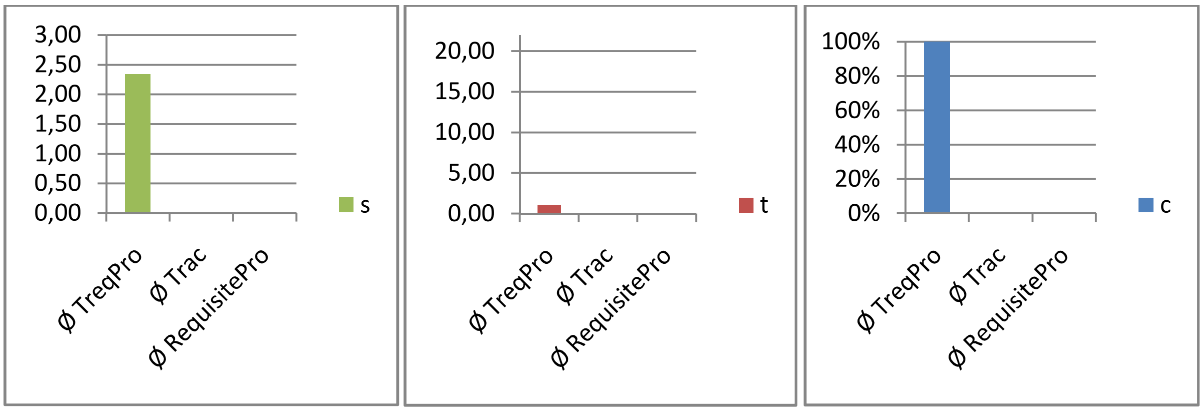


Figure 33: Average satisfaction (s), execution time (t) and completeness (c) for Task 9

5.2.10.2 Qualitative Feedback

TreqPro

- 2 users stated (separate from their rating in the 'satisfaction' field) that they liked the graphical navigation.

Trac

- 0 participants knew a possibility to get graphical navigation in Trac.

Requisite Pro

- 1 participant mentioned Requisite Pro's traceability tree (which has similar functionality as the graphical navigation in TreqPro) as well as jumping back and forth between requirements via the requirements' property windows.

6 Discussion and Further Work

In this chapter, I give an overview of the results and enhancements compared to existing work. I summarize quantitative results and the improvements of the prototype compared to 2 existing tools in the measured quantities in section 6.1. After that, I give an overview of qualitative results and the improvements identified in that field in 6.2. I line out possible further work in section 6.3.

6.1 Quantitative Results

Requirements Managers use standard software (Word Processor, Spreadsheets) or specific specialized tools for requirements management. As outlined in section 3, I have developed a new concept to address major shortcomings of existing tools and implemented this concept in the ReqM tool prototype *TreqPro*.

In the usability test with a set of common ReqM use cases, the prototype showed **considerable improvements** compared to the two existing Siemens standard tools **in all 3 measured parameters**:

6.1.1 User Satisfaction (s)

The participant satisfaction was significantly greater for TreqPro than for the two existing tools (+40 % compared to Trac, +61 % compared to Requisite Pro). For several tasks, less experienced users were considerably more satisfied with the existing tools than more experienced users. The greatest improvement in satisfaction could be reached in basic requirements operations. Both existing tools showed similar satisfaction.

6.1.2 Execution Time (t)

The execution time was significantly smaller for TreqPro than for the two existing tools (-34 % compared to Trac, -39 % compared to Requisite Pro). 3 of 5 compared tasks took the participants only half the time (40 – 50 %) with TreqPro compared to the other tools, while 2 compared tasks showed no significant change in execution time. The two existing tools showed similar execution time.

6.1.3 Completeness (c)

With 99 %, the completeness was significantly greater for TreqPro than for the two existing tools (+87 % compared to Trac with c = 53 % and +74 % compared to Requisite Pro with c = 57 %). While almost all tasks could be completed with TreqPro, some of the tasks could not be completed with the other tools because they do not provide the functionality:

- Project Template Configuration
- Document generation
- Semantic Tracing
- Graphical Navigation

6.2 Qualitative Results

Compared to existing tools, the new approach improved on the major shortcomings traceability, versioning and tool integration.

The **usability** of the prototype for basic requirements operations acquired extraordinary commendation. However, there is still room for improvement: The usability of display and creation of **traceability** is not optimal yet. Drag & Drop support for the navigation would make basic operations more intuitive.

On one hand, not all interviewed practitioners found full **versioning** support necessary. On the other hand, some of the experts expressed their desire for workflow support in requirements. This seems to depend largely on the target audience and a tool's "target project size".

Comparison of requirements and comparison of whole projects, though only implemented basically, clearly outperformed the **comparison capabilities** of the other tools.

The results prove that the general approach of building the ReqM tool as a web application fulfills user expectations in usability of the user interface, direct editing of requirements, and consistent collaboration on ReqM.

Generally, the approach of **graphical navigation** in requirements is very promising. The implementation included in the prototype is highly experimental and was only used for a simple show-case of easy navigation of relations across multiple artifacts. While a few frameworks for the navigation of graphs in web applications are available, considerable development effort would be necessary for a stable solution. As the results show, there is a high user acceptance of the graphical navigation. The impact of graphical navigation on usability and productivity of ReqM tools should be subject to further investigation.

The prototype can generate basic **requirements documents**. In contrast to the existing tools, it can generate a full specification. However, the prototype implementation is not configurable in any way, which is an important feature stressed by several test participants. The experts have proposed a templating system with individual configuration of each included element.

6.3 Further Work

From the usability test, a lot of feedback (defects and improvements) could be gained which is valuable input for the further development of the prototype. The prototype is not suitable for production use yet, but its concept proved to address the identified shortcomings. The great reduction in time and the increase in user satisfaction suggest a promising improvement of productivity when using an implementation of the concept in a production setting.

From a scientific point of view, I would suggest carrying out an **evaluation with an extended number of participants in a widened context** spanning multiple organizations and tools to learn more about the investigated aspects in comparison to other tools. Such an evaluation would also display an increased significance compared to the current study.

For this study, the implementation of an **improved prototype** based on the qualitative feedback gained through the usability tests would allow for even greater improvements compared to the existing tools.

From a practical point of view, it would be necessary to provide **better integration with development tools** in order to achieve full traceability throughout the development process. The Mylyn tool for example allows for integration of the Eclipse integrated development environment with the Trac ticket system and could be adapted to work with TreqPro with little effort. Improvement and **further development of the prototype** as an open-source project could finally lead to an open-source ReqM tool usable in a production setting.

Further necessary improvements for production use include (1) the integration of multimedial and arbitrary binary content, (2) the addition of workflows for better change management, (3) a rule engine for flexible customizable configuration of conditions inside the requirements model, (4) the implementation of suspect tracing, (5) full integration of the Trac permission system, (6) configurable document generation including relations. An implementation of the graphical navigation that goes beyond an experimental prototype would highly increase the usability of the system.

Last, but not least, the performance of the system should be improved to allow for high numbers of objects as encountered in typical projects.

7 Conclusion

Requirements Engineering (RE) is the economically most critical subfield of software development. Requirements Management tools help project managers and requirements engineers collaborate on requirements and keep track of requirements changes. Open-source tools are typically not inclined to limitations like high license and training costs, sophisticated use, limited integration with third-party tools or limited extensibility. This explains the general trend towards using open-source tools in software development. However, there are only few open-source ReqM tools, they are in poor state of development, and they lack of quality.

In this work, I have analyzed features of existing ReqM tools and discussed which features should be included in such tools (RQ1). Using scientific literature and holding interviews with ReqM experts, I have elaborated a comprehensive catalog of ReqM tool features, which I have evaluated in a web survey at Siemens Austria. The catalog proved to contain 97 % of the ReqM features considered important by the participating experts. The participants gave remarkably high priority to usability of the ReqM tool.

I have worked out three specific, major shortcomings of existing ReqM tools: (1) Limited versioning, (2) inadequate traceability support, and (3) strongly limited integration with other tools. I have proposed a concept to address these shortcomings (RQ2) and implemented this concept in the open-source ReqM tool prototype *TreqPro*. In a pilot study with 8 ReqM experts at Siemens Austria I have compared the prototype with two existing tools (the open-source solution Trac and the proprietary solution Requisite Pro) considering user satisfaction (s), execution time (t), and completeness (c) when executing a set of 9 standard ReqM use cases.

I have shown that the prototype displays considerable improvements in all 3 measured parameters: Satisfaction could be improved by 40 % compared to Trac and by 61 % compared to Requisite Pro. Execution time could be improved by 34 % compared to Trac and 39 % compared to Requisite Pro. Completeness could be improved by 87 % compared to Trac and 74 % compared to Requisite Pro.

The participants have given valuable qualitative feedback on the prototype which will be useful for further development of the application. To affirm the results found in the pilot study, I have proposed a broader investigation with multiple tools and an extended number of participants.

Appendix A: Catalog of ReqM Tool Features

A1 Setup, Customization, Administration and Usability

A1.1 Custom Requirement Types

- ❑ *Artifact* types (e. g. requirement types) can be customized (e. g. there can be requirements, stakeholder profiles, bug reports), see 0.

A1.2 User Administration

- ❑ Users with different permissions can be defined.
- ❑ Users can be organized in groups (like ‘project X’). The relationship type is m:n.
- ❑ Each group has a default role that new users have in this group.
- ❑ A role can have multiple permissions. The relationship type is m:n.
- ❑ Users can play roles (like ‘project manager’) in a certain group. The relationship type is ternary, i. e. a user’s role can be different for every group (for example, user A can be project manager in one group, but ordinary member in another).
- ❑ Permissions can be assigned to a role for objects based on a metadata attribute (e. g. to all objects with the value “use case” for the attribute “requirement type”, or to all objects with the value “new” for the attribute “state”).
- ❑ Permissions can be assigned to a role for objects in a path-based way (e. g. directory/package/component etc., similar to path-based subversion authorization).
- ❑ Import of large amounts of user data from external sources is supported.
- ❑ Integration of external authentication and authorization mechanisms (LDAP …) is supported.

A1.3 Usability

- ❑ Undo functionality is available.
- ❑ Search functionality is available.
- ❑ Drag & drop is available where appropriate.
- ❑ Shortcuts are available for often-used actions.
- ❑ Configurations on items in can be inherited in hierarchies, but overruled if necessary.
  - Administration: Configurations (e. g. for process types etc.) can be duplicated and reused.
  - User: Attributes and Relations can be inherited (e. g. X «is-in-charge-of» Z, Y is child of X, Y «is-in-charge-of» Z)
- ❑ State-of-the-art usability requirements have to followed, e. g. text box sizes.

A1.4 Adaptability and Extendibility of Tool Functionalities

- ❑ The menu items can be customized.
- ❑ Plug-ins can extend functionality.
- ❑ Scripts and macros can automate tasks.

A1.5 Online Help and Documentation

- ❑ An online help is available.
- ❑ Parts of the online help are available as context-sensitive help.
- ❑ Comprehensive tool documentation is available.
- ❑ Example projects are available.

A1.6 System Prerequisites

- ❑ Supported platforms and databases have to be specified exactly. (genau definieren)
- ❑ Memory, CPU, disk space necessary may not exceed capabilities of selected architecture.
- ❑ The highest possible amount of requirements managed by the tool has to be specified.

A1.7 Installation and Administration

- ❑ There is an easy-to-use administration frontend.
- ❑ The administration frontend protects users from errors by validating input.

A1.8 Scalability, Concurrency and Distribution

- ❑ Many projects can be managed in parallel (i. e. the tool supports opening more than one project at the same time).
- ❑ Users can work in these projects independently.
- ❑ Remote, concurrent, multi-user connections are supported.
- ❑ Concurrent editing of requirements is managed appropriately.
- ❑ AT LEAST ONE OF
  - ○ A rich client is available with full access to the functionality.

  OR
  - ○ A web client is available with full access to the functionality.

A1.9 Partial reuse of Project Settings and Data

- ❑ Configurations can be copied from one project to another.
- ❑ Data can be copied from one project to another.

A2 Capturing, Editing and Managing Requirements

A2.1 Flexible/Customizable Implementation of the desired RE Process

The desired *RE* process can be modeled inside the tool as a *process template*. The *RE* process defines the workflow and structure of the requirements data (requirements model).

- ❑ Workflows, Lifecycles, *Artifact* types, Associated Metadata types can be customized.
- ❑ Used Tools can be customized.

A2.2 Reusability of Process Templates

- ❑ The *RE* process (including the requirements model) defined in one project can be reused in another.
- ❑ The process remains adaptable for each project.
- ❑ *Process templates* can be duplicated and changed / built upon a master template.
- ❑ *Process templates* are versioned.

A2.3 Input and Import Methods for Requirements and Related Artifacts

There are various activities which require tool support:

- Requirements elicitation methods:
  - Workshops
  - Interviews
  - Questionnaires
  - …
- Other activities affecting requirements:
  - Bug Reporting
  - *Change Request*-> activity
  - …

All these activities have a **source**, which is either

- manual input from a stakeholder or
- automatically processed input from an existing *artifact* (which could also be an old version of the same *artifact*).

All these activities have one or more **resulting *artifacts*** each having (generally spoken)

- attributes (e. g. title, content, reporter, priority, owner, etc.) and
- relations (e. g. «is-related-to», «refines», «is-responsible-for») with other *artifacts*.

Examples for resulting *artifacts* are requirements, bug reports, stakeholder profiles, test cases, etc.

- ❑ Each resulting *artifact* is bidirectionally linked to its source (see traceability)
- ❑ Requirements elicitation methods are supported electronically (e. g. interview or questionnaire participants fill out an electronic form, workshop participants discuss in a forum, etc.).
- ❑ For the input data of each supported type of activity, adequate editors or adequate parsers are provided (Requirements could be recognized using tags (“REQ”) created by the user, structure predefinitions (Requirement Name, Description, Empty Line) or keywords (“.. shall ..”). Requirements IDs can be extracted from the document); i. e. the data can be entered into the system in a comfortable way.
  - If data can be entered manually (e. g. bug reports, *change requests*, interview answers), the forms for entering the data are configurable.
  - If data is parsed automatically, multiple sources can be defined.

    - If data is parsed from external sources, the parsed entities can be selected before import, e. g.: The system finds 24 requirements in 2 source documents. The user chooses to import 21 and to discard 3.
    - External documents (MS Office, etc.) can be used as sources. Ideally, there is a way to use the external tool (e. g. a text processor) interactively to edit the *artifacts*, e. g. by tagging certain paragraphs as requirements.
    - Each newly entered *artifact* (e. g. requirement, *change request*, questionnaire response, etc.) can have attributes and relations.
- ❑ Adequate views for each activity are supported; i. e. the acquired data can be browsed in a comfortable, meaningful way.
- ❑ Checklists help users avoid and identify characteristic, frequent errors (e. g. during review or interview/questionnaire preparation and execution).
- ❑ Response *artifacts* (e. g. questionnaire responses) can be analyzed automatically (e. g. by counting of check boxes).

A2.4 Artifact Representation

Artifacts (e. g. requirements, use cases, change requests, stakeholder profiles) can have attributes and relations, generally spoken. Attributes (e. g. “reporter”, “component”, “state”) are fields of metadata that structure the information for one specific *artifact*. They can be used as search criteria. Relations (e. g. «refines», «is-responsible-for») link different related *artifacts* to each other. Traceability is achieved through relations.

- ❑ Attributes can be configured independently for each type of *artifact* (e. g. most *artifacts* will have a text attribute, but a bug might also have a “severity” attribute).
- ❑ Attributes can have different types (String, Number, Boolean, …).
- ❑ Attributes can be mandatory or optional.
- ❑ Relations can be configured independently for each type of *artifact*.
- ❑ Relations have a name for each direction.
- ❑ Relations can be mandatory or optional.
- ❑ *Artifacts* and relations can be displayed as a graph.
- ❑ *Artifacts* and relations can be displayed as a hierarchical tree (relations: one at a time).
- ❑ *Artifacts* and relations can be displayed as a matrix.
- ❑ *Artifacts* may contain graphical models and mathematical expressions.
- ❑ *Artifacts* may contain arbitrary documents

A2.5 Artifact Categorization and Structuring

- ❑ Requirements can be categorized (e. g. by grouping of attributes or providing hierarchy views).

A2.6 Artifact Identification

- ❑ *Artifacts* can be identified by numbers as well as by symbols.
- ❑ *Artifacts* can be identified automatically or manually.
- ❑ IDs are unique across projects.
- ❑ IDs contain a project identifier.

A2.7 Artifact Reuse

- ❑ *Artifacts* can be transferred into other projects with few interactions.

A2.8 Artifact Query System

- ❑ *Artifacts* can be queried according to certain criteria (e. g. parts of attribute values, relations)
  - ○ Query results can be filtered according to these criteria.
  - ○ Query results can be sorted according to these criteria.
- ❑ A set of customizable, predefined queries is available.
- ❑ Ad-hoc (i. e. new, not predefined) queries are supported.

A2.9 Validation Support

- ❑ Checklists support the user for checking various criteria e. g.
  - ○ correctness
  - ○ completeness
  - ○ consistency
  - ○ verification
  - ○ comprehensibility
  - ○ clarity
  - ○ traceability
  - ○ modifiability
- ❑ The requirement description can be compared with a linguistic pattern (e. g.: “body must match 'The system shall .*'”).

A2.10 Artifact Prioritization Methods

- ❑ *Artifacts* (e. g. requirements) can be prioritized via assignment of a priority attribute.
- ❑ *Artifacts* can be prioritized via drag & drop.

A2.11 Description of Domain-Specific Terms

For the collection and clarification of domain-specific terms,

- ❑ AT LEAST ONE OF
    - ○ A glossary or domain knowledge database is available.

  OR

    - ○ Links to the relevant information are automatically created on domain-specific terms.

A2.12 Groupware Functionality

- ❑ Discussion forums are available.
- ❑ Wikis are available.
- ❑ Votings are available.
  - ○ Multiple stakeholders can assign a field value (e. g. priority) to each *artifact* (e. g. requirement).

    - An overall result is aggregated automatically.
    - Statistical values (mean, standard deviation ...) are available.
- ❑ Information can be structured to get an overview over related topics (e. g. categories, tags …)

A2.13 Journal Functionality

- ❑ Journal entries automatically create an overview on actions in the environment (who, when, what).
- ❑ The journal can be exported as a report.

A2.14 Modeling

- ❑ Multiple aspects of the desired system can be modeled in the tool (e. g. with UML or similar).
- ❑ Models created with the tool can be included in *artifacts* to help understand the requirements described there.
    - Models can be linked (allows for reuse) to the *artifacts*.
    - Models can be embedded in the *artifacts*.

A2.15 Notification on Requirements Change

- ❑ The owner / reporter of an *artifact* can be notified on change or state transition of the *artifact*.
- ❑ The owner can also be notified if related *artifacts* change, i. e. if any relation is “suspect”, see 0.

A2.16 Offline Editing of Artifacts.

Artifacts can be locked and edited offline. After merging the changes, the *artifacts* are unlocked automatically.

A3 Configuration Management Aspects

A3.1 Configurable Change Management Process

Depending on what processes are used, requirements may not be modified arbitrarily at all times. For example, the creator of a requirement might want to edit it to correct errors, but others should not be able to modify the requirement without formally issuing a *change request*.

It is quite common to allow “free” editing of requirements up to a certain date, when the features are “frozen” in a specification document and contracts are signed. All subsequent changes (not only to requirements, but also to other types of artifacts) have to pass the change management process.

When a requirement is changed, its relations to other requirements can be regarded as ‘suspect’, as the related requirements have to be validated / verified.

- ❑ All parts of the change process (like states and roles) can be configured.

A3.2 Artifact Versions and Branches

- ❑ *Artifacts* are versioned.
- ❑ *Versions* have a unique identifier.
- ❑ Multiple *versions* of an *artifact* can exist in parallel (branching).
- ❑ The tool supports working on different branches (e. g. with different databases).

A3.3 Artifact Baselines

- ❑ The state (including attributes and relations) of a certain set of *artifacts* in specific *versions* can be frozen (*baselining*).

A3.4 Comparison of Versions

- ❑ Different *versions* of *artifacts* can be compared.
- ❑ Deltas between these *versions* can be identified (e. g. through side-by-side displaying and highlighting).
- ❑ Requirements change packages can be identified.

A3.5 History of Artifacts

- ❑ The history of all *artifacts* can be retrieved.
- ❑ Creator and creation date can be retrieved for all *artifacts*.
- ❑ State transitions can be retrieved for all *artifacts*.

A4 Traceability of Requirements

A4.1 Traceability between Requirements and other Artifacts

Traceability between requirements allows for completeness of information about the creation of a requirement by linking to other *artifacts* that were included in the elicitation process. It supports change impact analysis, requirements validation, compliance verification, and regression test selection.

Basically, each *artifact* that is related to a requirement should be traceable to the other related *artifacts*. Each trace link should be available bi-directionally:

- ❑ Requirements can be traced to
  - ○ other requirements (e. g. traces between functional and relevant non-functional requirements)

  - interviews
  - stakeholders
  - questionnaires
  - questionnaire respondents
  - workshop documents
  - stakeholders (interviews, workshop documents)
  - change requests
  - review protocols (to identify who has checked a requirement at what date/time
  - system design components (in order to understand which design components contain which requirements)
  - the source code

  (and vice versa)

- ❑ Interview documents can be traced back to stakeholders (and vice versa).
- ❑ Questionnaire responses can be traced back to respondents (and vice versa).
- ❑ System design components can be traced to the source code (in order to understand which parts of source code implement which system design components) (and vice versa).

A4.2 Precision of Traces into Source Code

- ❑ Traces can be created at different precision levels.
- ❑ Traces can be done at method level.
- ❑ Traces can be done at class level.
- ❑ Traces can be done at component level.
- ❑ Traces can be done at *changeset* level.

A4.3 Change Impact Analysis

- ❑ The traces can be followed from one *artifact* to another.
- ❑ Inconsistencies between different *artifacts* can be identified.
- ❑ For a changing *artifact*, all traces from the changing *artifact* to related *artifacts*, are highlighted as suspect to indicate that the traced-to *artifacts* have to be checked for necessary adaptations.
- ❑ After the change has been implemented in these *artifacts*, the traces can be set on valid again manually.

A4.4 Support of Comprehensibility of a Trace

- ❑ It is possible to assign names to trace types, e. g. “tests” / “is tested by” for better comprehensibility of traces.

A4.5 M:N Relationships

- ❑ M:N traces are allowed (e. g. one requirement to many classes, or one class to many requirements).

A4.6 Manual Trace Generation

- ❑ Traces can be created manually by dragging from one *artifact* to another *artifact*.
- ❑ Traces can be created manually be adding a target *artifact* into a list in the source *artifact*.
- ❑ Traces can be created automatically by inserting a trace in a traceability matrix.

A4.7 Automated Trace Generation and Bidirectionality

Traces can be created automatically to reduce effort.

- ❑ AT LEAST ONE OF
  - ○ Automatic trace generation is done by parsing inserted requirements keys in source code files.

    OR
  - ○ Automatic trace generation is done by comparing text patterns in the requirements description.
- ❑ Bidirectionality is established automatically (by also creating the B-A reverse trace for each A-B trace created manually or automatically).

A4.8 Mandatory vs. Optional Traces

- ❑ A tracing policy allows the user to configure the necessity of traces.
- ❑ The creation of mandatory traces is enforced.

A4.9 Trace Representation

- ❑ Traces can be represented as a traceability graph with nodes and edges.
- ❑ Traces can be represented as a tree structure, e. g. with one requirement as the parent and related requirements as its children.
- ❑ Traces can be represented in traceability matrices.
- ❑ Starting from a single requirement, all related *artifacts* (other requirements, test cases, source code) can be accessed.

A4.10 Traceability across Tool Borders

Most *RE* tools offer adequate traceability between objects managed inside the tool, but poor, mostly unidirectional traceability regarding external sources (e. g. *RM* tools, modeling tools, IDEs, test management tools). Better, bidirectional traceability could be established

- by offering a good interface to other tools (e. g. a simple way to access each *artifact* managed inside the tool, like a URL in a web-based tool) as well as
- by offering user-defined, searchable fields and / or links to access *artifacts* managed in other tools.

- ❑ There is a simple way to access each *artifact* managed inside the tool and link to it.
- ❑ Each type of *artifact* managed inside the tool can have user-defined fields.
- ❑ There is a simple way to search and filter for specific values of user-defined fields.

- ❑ There is a synchronization mechanism that highlights changes each tool if something changes in the other.

A4.11 Traceability between Projects

- ❑ *Artifacts* in one project can be related to *artifacts* in a different project.

A5 Document and Report Generation

A5.1 Report Generation

Reporting is useful for gaining information about the project status, presenting results to stakeholders and generating specification documents. Some commonly used reports are:

- Traceability Reports: These reports are used to provide an overview about inter-artifact dependencies.
- Diff Reports: These reports are used to compare two artifacts to ease verification of the changes.
- Requirements Coverage Reports: These reports are used to assess the coverage of requirements with test cases.
- Requirements Documents: These documents consist of a structured overview of requirements.

It is often important to allow reporting across tool borders, i. e. to access external data sources from the reporting tool.

- ❑ EITHER
  - ○ The tool contains adequate reporting functionality.
    OR
  - ○ The tool provides an adequate interface to a reporting solution (e. g. BIRT, SODA).
- ❑ Various types of reports about the requirements can be generated (requirements per stakeholders, requirements which satisfy defined criteria, reports about traces / suspected links, status report about the compliance of requirements).
- ❑ The data (attributes, relations) included in the report can be freely chosen:
  - ○ The data can be filtered by specific attribute values like ‘priority == high’ or relations like ‘«is-responsible» for requirement X’.
  - ○ The artifacts to display can be selected in the user interface (e. g. a subset of all requirements).
- ❑ Reports can contain statistical values (sum, average …) and simple calculations to provide metrics for the project status.

A5.2 WYSIWYG Editor

- ❑ A WYSIWYG editor allows for preview on the generated reports.

A5.3 Charts and Graphs in Reports

- ❑ Artifact data can be displayed in diagrams, graphs, and data charts.
- ❑ The graphical output can be sorted and filtered, see 0.

A5.4 Report Formatting

- ❑ The layout can be defined using templates specifying pages, headers, footers …
- ❑ Templates can be imported from an external source in addition to defining them in the tool.

A5.5 Report Document Formats

- ❑ Reports and documents can be created in PDF format.
- ❑ Reports and documents can be created as MS Office documents.

A6 Interfaces to other Tools

A6.1 Integration with Development Environments

The integration of IDEs allows for direct linking of source code to artifacts. The *RE* tool must provide an adequate interface.

- ❑ Development Environments (Eclipse, NetBeans, VisualStudio …) is/can be integrated.

A6.2 Integration with Configuration Management Tools / Systems

- ❑ A Source Management tool (Subversion, CVS, SourceSafe ...) is/can be integrated.

A6.3 Integration with Test Management Tools / Systems

- ❑ EITHER
    - ○ The tool provides functionality for test case management.

      OR

    - ○ A TM tool is integrated to manage test cases and link them to requirements.

A6.4 Long Term Archiving Functionality

- ❑ Environment, resources and documents can be archived in a format that allows reactivation of the project or at least retrieval of relevant information.

A6.5 Interfaces

- ❑ The database is accessible via ODBC.
- ❑ Data exchange via XML is possible.

A7 Costs

A7.1 Adequate Cost-benefit Ratio

The cost-benefit ratio (considering license and training costs) of the tool must be acceptable.

List of Tables

List of Figures

References

[Bor10] Borland. CaliberRM Homepage. Available at: http://www.borland.com/us/products/caliber/index.html. Accessed August 25, 2010.

[Fin00] Finkelstein A, Emmerich W. The future of requirements management tools. Paper presented at: in Quirchmayr, G., Wagner R., and Wimmer M. (Eds.) (2000): Information Systems in Public Administration and Law. Österreichische Computer Gesellschaft, 2000.

[For04] Barnett L. Applying Open Source Processes In Corporate Development Organizations (Forrester, Inc.). *Best Practices*. May 2004.

[For06] Schwaber C. The Changing Face Of Application Lifecycle Management (Forrester, Inc.). August 2006.

[Fow03] Fowler M. Active Record. *Patterns of enterprise application architecture*: Addison-Wesley; 2003:160ff.

[Gee10] Geensoft. Reqtify Homepage. Available at: http://www.geensoft.com/.

[Got94] Gotel OCZ, Finkelstein CW. An analysis of the requirements traceability problem. Paper presented at: Requirements Engineering, 1994., Proceedings of the First International Conference on, 1994.

[Hei06] Heindl M, Reinisch F, Biffl S, Egyed A. Value-Based Selection of Requirements Engineering Tool Support. Paper presented at: EUROMICRO '06: Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006.

[Hei10] Heise Mobile. Press Report "Android is 2nd in Wireless Operating Systems" (German). Available at: http://www.heise.de/newsticker/meldung/Google-Android-auf-Platz-2-der-Wireless-Betriebssysteme-1077483.html. Accessed September 16, 2010.

[Hof04] Hoffmann M, Kuhn N, Weber M, Bittner M. Requirements for requirements management tools. Paper presented at: Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International, 2004.

[IBM10] IBM. Rational Requisite Pro Homepage. Available at:

http://www.ibm.com/software/awdtools/reqpro/. Accessed August 25, 2010.

[IEE94] IEEE recommended practice for software requirements specifications. Paper presented at: IEEE Std 830-1993, 1994.

[INC10] INCOSE Tool Survey. Available at: http://www.incose.org/ProductsPubs/products/rmsurvey.aspx. Accessed August 12, 2010.

[Kot98] Kotonya, Sommerville. *Requirements Engineering*. West Sussex: Wiley; 1998.

[Kre06] Kretzl B. Diplomarbeit

[Lai09] Lai E. Linux's share of netbooks surging, not sagging, says analyst - computerworld.com. November 4, 2009. Available at: http://www.computerworld.com/s/article/9140343/Linux_s_share_of_netbooks_surging_not_sagging_says_analyst. Accessed September 16, 2010.

[Lef03] Leffingwell, Widrig. *Managing Software Requirements - Second Edition - A Use Case Approach*. Boston, MA 02116: Pearson Education, Inc.; 2003.

[Lud10] Ludwig Consulting Services L. Requirements Management Tools. *Managing Requirements*. Available at: http://www.jiludwig.com/Requirements_Management_Tools.html. Accessed August 13, 2010.

[MKS10] MKS. DOORs Homepage. Available at: http://www.mks.com/solutions/discipline/rm/requirements-management. Accessed August 26, 2010.

[NAS10] Market NS. Directory of Requirements Management Software. Available at: http://software.nasdaq.com/requirements-management-software. Accessed August 12, 1010.

[Net10] Netcraft Ltd. January 2010 Web Server Survey. Available at: http://news.netcraft.com/archives/2010/01/. Accessed September 16, 2010.

[Ost10] SourceForge Project Page for OSRMT. Available at: http://sourceforge.net/projects/osrmt/. Accessed August 12, 2010.

[Pyt10] Python. Available at: http://www.python.org/. Accessed August 12, 2010.

[Ram01] Ramesh B, Jarke M. Toward Reference Models for Requirements Traceability. *IEEE Trans. Softw. Eng.*. 2001:58-93.

[Ram97] Ramesh B, Stubbs C, Powers T, Edwards M. Requirements traceability: Theory and practice. *Ann. Softw. Eng.*. 1997;3:397-415.

[Red10] Lang JP. Redmine Project Website. Available at: http://www.redmine.org/. Accessed August 12, 2010.

[Rei09] Reinisch F. Presentation at Subconf on Oct 29, 2009

[Som97] Sommerville, Sawyer. *Requirements Engineering - A Good Practice Guide*. West Sussex: Wiley; 1997.

[Spa05] Spanoudakis G, Zisman A. Software Traceability: A Roadmap. *Handbook of Software Engineering and Knowledge Engineering*. 2005;Vol. III: Recent Advancements.

[Spa10] Systems S. Enterprise Architect Homepage. Available at: http://www.sparxsystems.com/products/ea/index.html. Accessed August 26, 2010.

[Sub10] CollabNet. Subversion Home Page. Available at: http://subversion.tigris.org/. Accessed August 26, 2010.

[Tot07] Totz G. Pluginbasiertes Requirements-Tracing in der Softwareentwicklung

[Tra10] Trac Project Website (Edgewall Software). Available at: http://trac.edgewall.org/. Accessed August 12, 2010.

[Uen08] Uenalan O, Riegel N, Weber S, Doerr J. Using enhanced wiki-based solutions for managing requirements. Paper presented at: Managing Requirements Knowledge. MARK '08., 2008.

[Vis02] Freimut B, Punter T, Biffl S, Ciolkowski M. *State-of-the-Art in Empirical Studies*: Virtuelles Software Engineering Kompetenzzentrum (VISEK); 2002. ViSEK/007/E.

[Wik10b] Wikipedia-En. Usage share of operating systems. Available at: http://en.wikipedia.org/w/index.php?title=Usage_share_of_operating_systems&oldid=352719161. Accessed March 29, 2010.