A Case Study on Value-based Requirements Tracing

Matthias Heindl Stefan Biffl
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{heindl, biffl}@qse.ifs.tuwien.ac.at

ABSTRACT

Project managers aim at keeping track of interdependencies between various artifacts of the software development lifecycle, to find out potential requirements conflicts, to better understand the impact of change requests, and to fulfill process quality standards, such as CMMI requirements. While there are many methods and techniques on how to technically store requirements traces, the economic issues of dealing with requirements tracing complexity remain open. In practice tracing is typically not an explicit systematic process, but occurs rather ad hoc with considerable hidden tracing-related quality costs. This paper reports a case study on value-based requirements tracing (VBRT) that systematically supports project managers in tailoring requirements tracing precision and effort based on the parameters stakeholder value, requirements risk/volatility, and tracing costs. Main results of the case study were: (a) VBRT took around 35% effort of full requirements tracing; (b) more risky or volatile requirements warranted more detailed tracing because of their higher change probability.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *software quality management (SQA*)

General Terms

Management, Measurement, Documentation, Economics, Experimentation, Verification.

Keywords

Requirements tracing, value-based software engineering, case study, empirical evaluation.

1. INTRODUCTION

Software development includes the production of various types of artifacts, e.g., requirements specification documents, architecture descriptions, source code, and test cases. These artifacts provide different views on the system at different points of time. It is obvious that these artifacts do not exist in isolation from each other. Instead, they are related to and affect each other, e.g., if one requirement in the requirements specification document changes, other documents often have to be changed in order to preserve consistency. Furthermore, these artifacts typically evolve to some extent concurrently during development.

Requirements tracing is the ability to follow the life of a requirement in a forward and backward direction [8]. In the software development context, requirements tracing has important benefits, e.g., capturing traces in weekly intervals during development can support developer teams in keeping an overview on which requirement is implemented where in the source code. Project managers aim at keeping track of interdependencies between various artifacts of the software development lifecycle to find out potential requirements conflicts, to better understand the impact of change requests, and to fulfill process quality standards, such as CMMI requirements [17].

In literature approaches like [4, 12, 18] support requirements tracing activities like identification of requirement conflicts, change management and impact analysis, release planning, program comprehension, model consistency checking, and testing (verification and validation). However, identifying and maintaining trace dependencies leads to additional effort that can get prohibitively expensive with increasing number of requirements and increasing tracing precision. In practice, tracing is typically not an explicit systematic process, but occurs rather ad hoc with considerable hidden tracing-related quality costs.

Methods, tools and approaches of requirements tracing reported in literature [13, 20] provide technical models about how to store identified traces. Most tools aim to automate requirements tracing, but tracing automation is still complex and error prone. Furthermore, automation alone cannot really reduce efforts of requirements tracing. Thus, requirements tracing may seem too costly for routine use in practice. A major reason is that existing approaches make no difference between requirements that are very valuable to trace and requirements that are much less valuable. Tracing value depends on parameters like stakeholder importance, risk or volatility of the requirement, and the necessary tracing costs. Thus, there is the need for requirements tracing approaches that take these parameters into consideration, such as value-based requirements tracing (VBRT).

Systematic full tracing, where every requirement is traced with the same precision independent of its value, provides benefits in saving time in implementing error reports or change requests after the project has been finished. In this case the costs for newcoming maintenance personnel to re-discover knowledge about interdependencies in the system would usually be much higher than for identifying and storing trace dependencies during development with the original developers present. It seems easier to identify traces during development than later after project completion when a change request occurs.

Capturing all requirements traces can get complex and expensive very fast, e.g., imagine a software development project with

ESEC-FSE'05, September 5–9, 2005, Lisbon, Portugal.

only 18 requirements but more than 4000 traces to store and maintain. In comparison to full tracing, VBRT promises in a typical project significant reduction of tracing costs without losing its benefits.

The VBRT approach provides a technical model and an economic model for requirements tracing, depending on criteria like number of requirements, value of requirements, risk of requirements, number of artifacts, number of traces, precision of traces, size of artifacts, cost/effort of trace identification and maintenance, and value of traces. Figure 1 illustrates that traces can have different levels of precision, e.g., traces in code at method, class, or package level. Traces exist between all kind of artifacts, e.g. design, code, test cases. Thus, VBRT can help to perform cost-efficient requirements tracing within given budget limitations.

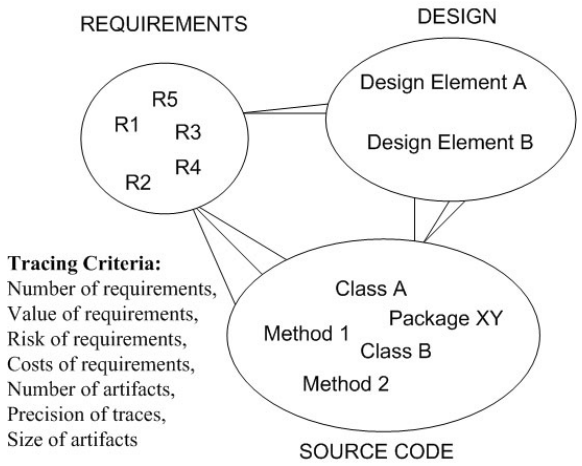

Figure 1. Requirements Tracing Overview.

Project stakeholders like customers, project managers, and quality managers do not put equal value on each requirement. Value-based software engineering approaches such as release planning [21] relate value differences to project decisions and practice. Similar to the requirements themselves requirement traces are not equally important. However, existing approaches treat each trace the same way and do not consider these value differences. With limited resources the project team has to decide capturing which traces seems most worthwhile.

The VBRT approach consists of 5 steps: requirements definition, requirements prioritization, requirements packaging, requirements linking, and evaluation. VBRT reduces tracing efforts by prioritizing requirements. Requirements prioritization uses the input parameters stakeholder value, requirements risk/volatility, and tracing costs to decide which requirements are valuable enough to trace and which are not.

A main contribution of this paper is a case study where we applied VBRT to a real-life project and report initial case study results, e.g., tracing-related costs. The focus of this paper was on traces between requirements and other artifacts, especially code pieces (vertical traceability), while horizontal traceability is part of further work.

In this paper we evaluated the VBRT approach in a real-life project setting and compared costs and benefits of VBRT to ad hoc tracing and full tracing. The case study results suggest that VBRT can be an attractive tracing alternative for typical software development projects, because it provides "traditional" benefits of tracing and minimizes tracing efforts at the same time.

The remainder of the paper is organized as follows: section 2 summarizes related work on requirements tracing; section 3 introduces the VBRT process and formulates research questions. Section 4 outlines the case study and presents results. Section 5 discusses these results. Finally section 6 concludes and describes further work.

2. RELATED WORK

This section presents related work on motivation for requirements tracing in software quality standards, need for tracing in practice, and on overview on traceability techniques as context of value-based requirements tracing.

2.1 Requirements Tracing in Quality Standards

There are many formal definitions of requirements in literature, e.g., Karlsson [14] defines a requirement as a current or future need that may be fulfilled. Requirements traceability can improve system quality, because it makes knowledge about the designed system independent from people, e.g., from the development team that has originally implemented the system, and therefore eases maintenance and change request implementations.

Many standards for systems development such as the U.S. Department of Defense (DoD) standard 2167A mandate requirements traceability practice. [19]. Requirements traceability is an issue for an organization to reach CMM level 3. Most organizations work on CMM level 1 or 2. In an assessment for reaching maturity level 3 there are questions concerning requirements tracing: whether requirements traces are applied to design and code and whether requirements traces are used in the test phases.

2.2 Need for Requirements Tracing

Two characteristics of software projects lead to the necessity of requirements traceability:

First, customers usually define requirements at the beginning of a software project. Therefore, a project manager has to use verification and validation (V&V) approaches, e.g., acceptance testing [11], throughout the project in order to ensure that the developed artefacts and products meet these customer requirements.

Second, changing requirements, rather than stable ones, are the norm in systems development [10]. Requirements are hardly ever stable, because the customers' situation may change and that is why also requirements may change during or after a software project. That means that requirements are prone to changes. Since many artefacts come up during a software project, a requirement change may have effects on many different artefacts.

The need for verification and validation and the instability of requirements force to trace requirements somehow. For instance, there are two situations, where some form of requirements tracing (RT) is essential:

(1) *Acceptance testing:* At the end of the software project the customers and the project manager perform a V&V step where

they have to decide if the developed software system meets the requirements initially defined by the customers. Usually they make this decision by performing an acceptance test where the functions and the external behaviour of the software product are compared against the specified requirements.

(2) *Change request:* Very often initially defined requirements change during the development process and even change after the project has been finished. Usually, the customer proposes such changes in the form of change requests or error reports. The project manager then has to decide if the proposed change request should be implemented or not. Therefore, he has to compare efforts and costs necessary to implement a change request against the value generated by change request implementation. In other words, he has to understand the technical impact of a change to be able to estimate change effort.

For both kinds of situations it is helpful to have knowledge about dependencies between requirements and all kinds of work products. This knowledge, as provided by requirements traceability (RT) techniques, can support the project manager in making decisions like the ones described above and can help increase system quality.

Current literature contains ample publications about the need for traceability, e.g., in Gotel and Finkelstein [8]. Watkins and Neal [22] report how requirements traceability aids project managers in: accountability, verification, consistency checking of models, identification of conflicting requirements, change management and maintenance, and cost reduction.

Accountability. During internal and external audits, the project will have a better success rate if the data is available for auditors and you can prove that a requirement was successfully validated by associated test cases [22]. Project managers have a better handle on costs, and customers are assured of getting the product they requested. Thus customer confidence and satisfaction is enhanced.

Verification. RT helps to verify that software requirements have been allocated both to design and code and to test cases and procedures for verification. This ensures that only required functions are designed into the product; as well as that all requirements have associated design components and qualifications test cases. Traceability ensures customer satisfaction by providing a documented means by which to prove to the customer that all of the stated requirements are met and that the job is completed [19]. By using traceability to acceptance test plans for every validated requirement, including derived requirements, the project manager can prove to the customer that the system completely meets their needs [19].

Consistency checking of models. Different models, e.g., use cases, state charts, sequence diagrams, and class diagrams, represent a set of perspectives on a software system. Since requirements traces store information about dependencies between different artifacts or models, and requirements, requirements traceability can provide support in checking consistency of a given set of models [4].

Identification of conflicting requirements. Very often requirements affect each other, e.g., a quality requirement affects a functional requirement. In other cases requirements do not only affect each other, they even conflict. By means of requirements traceability approaches, e.g., see Egyed and Grünbacher [5, 6], these cooperating or conflicting requirements can be identified.

Change Management/Maintenance. For each change, it is easier to determine what related elements of the design are affected. This helps to keep documentation up to date as the implementation progresses. In addition, managers can identify test procedures that should be re-run in order to verify the change. This knowledge helps save test resources. There are many papers about how to manage change within development processes and how to determine impacts of changes, e.g., Harker and Eason [10].

Cost reduction. Traceability allows allocating product requirements early in the development lifecycle. The cost of waiting until the integration and system test phase to correct defects (in untraceable components) may be as much as 30 times higher than defect correction in the earlier development phases [1].

2.3 Traceability Techniques and Approaches

Gotel and Finkelstein mention some basic techniques for RT, namely cross referencing schemes [7], key phrase dependencies [12], templates, RT matrices, hypertext [13], integration documents [15], and constraint networks [3]. These techniques differ in the quantity and diversity of information they can trace between, in the number of interconnections between information they can control, and in the extent to which they can maintain requirements traces when faced with ongoing changes to requirements. There are also tools to semi-automate requirements tracing [4, 18].

These techniques provide technical support to perform requirements tracing, but do not consider value and cost as argued in value-based software engineering [1, 2].

3. VALUE-BASED REQUIREMENTS TRACING (VBRT)

The goal of the value-based requirements tracing process is to identify traces based on prioritized requirements and thus to identify which traces are more important and valuable than others. The following subsections provide a VBRT process overview, simple cost-benefit model, and research question.

3.1 VBRT Process Overview

Figure 2 depicts process activities, actors, and deliverables of VBRT. In an iterative life cycle the VBRT process represents one cycle of developing and refining the value-based traceability system.

The VBRT process consists of five distinct steps: (1) requirements definition, (2) requirements prioritization, (3) packaging of requirements, (4) linking of artifacts, and (5) evaluation.

During *(1) requirement definition* the project manager or requirements engineer analyzes the software requirements specification and identifies atomic requirements. The requirements engineer then assigns a unique identifier to every requirement. The result is a list of requirements and their IDs.

During *(2) requirements prioritization* all stakeholders assess the requirements and estimate the value, risk, and effort of each requirement. The result of this step is an ordered list of requirements where the requirements are ranked on three priority levels [16].

(3) Requirements packaging is an optional process step that allows a group of architects to identify clusters of requirements. These clusters are needed to develop and refine an architecture from a given set of requirements. Grünbacher and Egyed [9] provide an overview on a generic intermediate architecture model based on requirements.

During *(4) requirements linking* the project team establishes traceability links between requirements and artifacts. Important requirements are traced in more detail than less important requirements. Therefore, we use 3 levels of tracing intensity. The result of this step is an overall traceability plan.

During (5) *Evaluation* the project manager can uses traces for certain purposes, e.g., to estimate the impact of change for certain requirements.

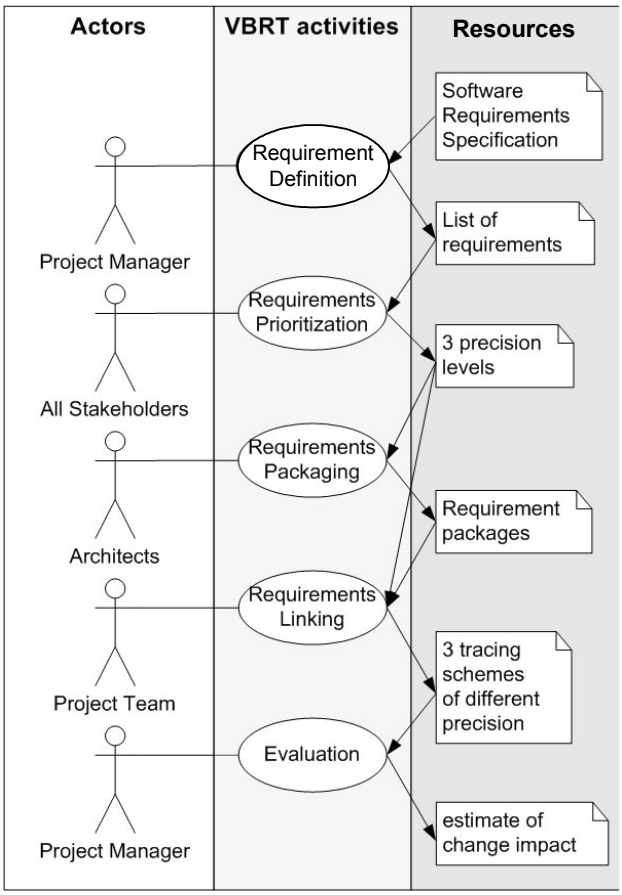

Figure 2. VBRT Process Overview.

3.2 Cost-benefit Model for VBRT

Tracing techniques typically provide only technical support to perform requirements tracing but do not take value and cost considerations into account.

Complete tracing needs often prohibitive effort and duration in a software project. The question arises, how much effort for tracing is appropriate to provide significant savings during usage of traces in a project. We want to optimize the cost-benefit of tracing and trace analyses. We assume that value-based requirements tracing can help to find a subset of traces that saves proportionally more cost than it loses benefit.

The costs and benefits of requirements tracing depend on the following parameters.

Project context:

- *Number of artifacts* to be traced; the higher the number of artifacts, the higher is the effort to create traces between them. It depends on the trace applications (e.g., requirements conflicts identification, change impact analysis, consistency checking, verification) which artifacts should be considered for tracing.
- *Number of requirements* in a software development project; Due to n^2 complexity of requirements tracing (potential traces between all n artifacts), the effort explodes with increasing number of requirements. That is one of the main problems of requirements traceability. VBRT is an approach to get a grip on the tracing effort problem.
- *Value of requirements* that is the importance of each requirement to the stakeholders (e.g., on a three-point-scale); We suppose that high-value requirements need more detailed tracing than low-value requirements, because they represent the core functionality of the system and trace information of the latter is more important than trace information of low-value requirements.
- *Risk of requirements*, that is the volatility of each requirement (e.g., on a three-point-scale); It seems to be worthwhile to trace risky/volatile requirements in more detail, because during trace applications like change impact analysis, these traces are needed more frequently than traces to stable requirements.

Tailoring parameters:

- *Number of traces*; the higher the number of requirements and artifacts to be traced, the higher is the number of potential traces to identify and maintain;
- *Precision of traces*, e.g., traces between requirements and code could be at lines of code level, method level, class level, or package level [4, 5];
- *Complexity/Size* of trace objects, e.g., if a code class is extremely big, then tracing at method level would provide a higher value than tracing at class level. If the class is very small and contains only one method, then tracing at class level provides nearly the same value as tracing at method level.

Cost and benefit:

- *Cost/Effort for tracing*, e.g., more precise traces are more expensive to identify than less precise ones. Reducing the number of traces, e.g., by omitting tracing of less important requirements, also has an effect on costs and efforts.
- *Value of traces* in context of a specific application, e.g., change impact analysis, identification of requirements conflicts, etc. For example, in context of change impact analysis, using traces reduces costs and time for locating code pieces to be changed.

We compare three tracing alternatives in this paper. The first alternative is *ad hoc tracing*. The project team does not create and maintain any kind of traces during development, but

searches documentation for relationships when needed. This variant has hidden efforts for search and rework risk.

The second alternative is *full tracing*. The project team does not make differences between requirements and traces each requirement with the same effort and precision. It makes a difference in this variant whether the project team identifies traces during development or after the project (ex post). The latter approach seems to be considerably more expensive than the first, as the project team has often to re-discover system details.

As described above, full tracing provides certain benefits in comparison with ad hoc tracing, but there is still a potential for improvement or optimization, e.g., full tracing wastes efforts for tracing requirements that do not really need to be traced with that level of precision. Thus, the criterion for optimization is which requirements should be traced at which level of precision.

VBRT addresses this issue by providing a requirements prioritization step where requirements are assigned to one of three precision levels. For example, a ratio of requirements per precision level of 10%:30%:70% or 20%:40%:40% would provide a considerable effort reduction. Thus, VBRT tailors tracing efforts down to manageable size without losing too much of the benefits of full tracing.

The scope and prioritization of requirements is important as not all requirements do have similar value and trace sets are usually not complete due to effort constraints and duration of trace creation in the software development process. Therefore, the question arises which traces are most worthwhile to create and maintain in a software project.

One aspect is the value of a trace set for the stakeholders, another is the risk of change volatility of a trace, and finally the cost of creating and maintaining traces. We performed a requirements prioritization step in our case study based on the prioritization approach by Ngo-The and Ruhe [16] in order to identify most important requirements, medium important requirements and less important requirements.

3.3 Research Question

In context with the VBRT process, this work deals with the following research question:

- RQ: To what extent can VBRT reduce requirements tracing efforts (economy of requirements tracing)?

We assume that VBRT reduces tracing efforts by omitting identification of unimportant traces through requirements prioritization. We measure the tracing effort in person hours to evaluate this research question.

We assume that traces differ in their value depending on requirements' value, costs, and risks (volatility). We evaluate this question by analyzing the results of the prioritization step, where requirements are assigned to precision levels. The most valuable, most costly, and most risky (volatile) requirements should be traced on the highest precision level.

To gather data to answer this research question, we performed a case study at Siemens Austria. Focus of the case study was to apply the VBRT process in a small project that allows comparing full tracing and VBRT effort and discussion of empirical data with development experts. The case study should be a basis for extrapolation of tracing and cost-benefit parameters to a typical larger project.

4. CASE STUDY APPLICATION OF VBRT

The case study project "public transport on demand" is about an improved and more efficient public transportation system in rural areas supported with modern information technologies. The challenge is to stop further deterioration of public transportation access in rural areas with a new traffic model. The basic element in the system is a public transportation service provider centre (PTSPC). The passenger can ask the PTSPC via SMS, Internet or Call Center for transportation on a route within the service area. The passenger has to provide input parameters like starting point, destination, arrival or departure time, maximal amount of transfers, and maximal acceptable travel time. If the location of the start or the destination has no scheduled stop within walking distance, the system will arrange a feeder service to or from the stop. Passengers can ask the PTSPC for route information and prices; they can also directly buy their tickets. The PTSPC is thus able to calculate the best possible route and the price of the requested trip. Figure 3 illustrates the target traffic communication model: A customer orders a route with his handy via call center, the PTSPC calculates a route consisting of transport mode options, stops, and pickup times. Finally, the customer receives a SMS with the route information.

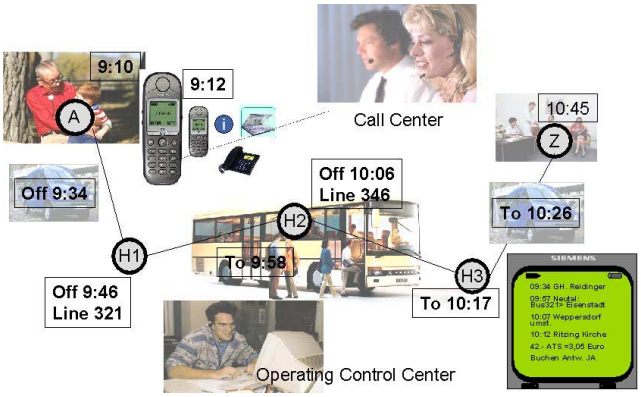

Figure 3. Communication example from case study project.

The PTSPC ensures that all orders are executed properly. The PTSPC also arranges for a follow-up acknowledgement between the feeder system and the buses. All vehicles of the feeder service and all the buses are equipped with location and communication devices. Therefore, the PTSPC knows the locations of all the vehicles in the system and can communicate with their drivers. The PTSPC operator is thus able to notify the drivers and arrange for appropriate actions to be taken in the case of major unexpected deviations from schedule.

The type and size of the project was suitable for us to apply the VBRT approach in a software development project with realistic yet manageable trace options. The project consisted of 46 requirements, which seemed to be the right magnitude to evaluate the VBRT approach, because the number of requirements was neither too high nor too little. The following artifacts existed when we started the case study:

- *Software requirements specification:* The specification contained the description of functional requirements. Non-functional requirements, e.g., quality, performance, reliabil-

ity, were not described and therefore excluded from our case study.

- *Architecture description and high-level design:* These artifacts described the building blocks of the desired system.
- *Prototype:* The prototype was a partial implementation of the requirements in the software requirements specification.

The following subsections describe the VBRT process steps in context of the case study.

4.1 Requirements Definition

The prerequisite for requirements definition is a software requirements specification. This software requirements specification is written in plain text and most of the functional requirements are modeled as use cases. The software requirements specification contains 46 functional requirements. The main task of this process step for the investigator was to review the software requirements specification and to extract each use case title into an excel list.

Results of requirements definition

One person needed approximately 3.5 hours to generate the *requirements list* from a textual software requirement specification.

4.2 Requirements Prioritization

When we performed the case study the project team consisted of three project members: the project manager, the quality manager, and one additional project member. These three persons performed the prioritization step. The project manager had to assess the value, risk, and effort of each requirement. All other project members had to assess the value only (stakeholder value proposition). Table 1 illustrates a part of the project manager's prioritization sheet and table 2 depicts a part of the standard prioritization sheet for all other project members. In order to support the understanding of each requirement, the working sheets contained short descriptions of every requirement and every requirement description contained a link to the relevant chapter of the software requirements specification. The project manager assessed value, risk, and costs of each requirement, whereas the rest of the project team assessed just the value of each requirement (on a scale with high importance +, medium importance 0, and low importance -).

Table 1. Prioritization sheet for the project manager

Requirements	Value	Risk	Effort
Req1. Registration, Login, Logout	-	-	0
Req2. Change user profile via internet	0	0	0
Req3. Order a route via internet	+	+	+
Req4. Pre-configure a route	+	0	0
Req5. Order a route via SMS	+	-	0
Req6. Order a route via call center	+	+	+

Table 2. Prioritization sheet for all other project members

Requirements	Value
Req1. Registration, Login, Logout	+
Req2. Change user profile via internet	0
Req3. Order a route via internet	+
Req4. Pre-configure a route	+
Req5. Order a route via SMS	+
Req6. Order a route via call center	+

Based on the three project members' assessments, we calculated a general result table. We counted the number of +, 0 and - from the value assessment. Based on these counts, we calculated the overall stakeholder value classification, SV (see table 3).

The classifications of risk R and effort E, performed by the project manager, resulted in the classification RE, reflecting the overall risk/effort situation for every requirement, ranging from '--' very low to '++' very high. Both the stakeholder value classification, SV, and the risk/effort situation, RE, were input to determine a priority level L, ranging from 1 – high priority – to 3 – low priority (see table 3). The prioritization approach is described in [16] in detail.

Results of requirements prioritization

The project manager assessed value, risk, and effort for each requirement, whereas other project members assessed only the value. The duration of the prioritization step took per person between 40 and 60 minutes. The assessing project members did not have to cooperate but performed their assessment individually. Table 3 depicts the overall assessment of the requirements list.

Table 3. Stakeholder requirements prioritization results

List of requirements	Value			R	E	RE	SV	L
	+	0	-					
Req1. Registration, Login, Logout	2	0	1	-	0	-	+	2
Req2. Change user profile via internet	0	3	0	0	0	0	0	2
Req3. Order a route via internet	2	1	0	+	+	++	+	1
Req4. Pre-configure a route	2	0	1	0	0	0	+	2
Req5. Order a route via SMS	2	1	0	-	0	-	+	2
Req6. Order a route via call center	3	0	0	+	+	++	+	1
Req7. Change the user profile via call center	1	2	0	-	-	--	0	3
Req8. Administration of orders via internet	3	0	0	+	+	++	+	1
Req9. The driver may see order details of his orders	2	1	0	0	0	0	+	2
Req10. Data transfer between taxis and the central dispatcher	1	2	0	-	+	0	0	2

The columns value +, 0, and - contain the number of votes, e.g., 2 project members voted for Req1 to be important (+), and one project member voted for Req1 to be unimportant (-). The columns R and E contain the project manager's assessment of risk

and efforts for each requirement. The column RE contains the combination of R and E; column S contains the combination of the value assessments. Finally, column L contains the assignment of each requirement to one of three priority levels, e.g., Req1 is assigned to level 2 (medium importance) and Req3 is assigned to level 1 (high importance). The prioritization step is described in Ruhe [16] in detail.

The VBRT prioritization step has the following characteristics:

Prioritization is short. In the case study the average duration for the stakeholders' prioritization was 50 minutes. The number of requirements in each priority level seems to be suitable. The distribution of requirements to the three priority levels was approximately 1:6:2, that means out of 9 requirements 1 requirement was priority level 1, 6 requirements were priority level 2, and 2 requirements were priority level 3.

4.3 Requirements Packaging

After the project team classified each requirement on one priority level, the next optional step is about generating an architecture proposal by means of an intermediate model [9]. This requirements packaging step is not described in detail because it is optional and not directly within the focus of this paper.

4.4 Requirements Linking

Requirements on priority level 1 are traced in more detail than requirements on priority levels 2 and 3. This graduation of intensity reduces the overhead for tracing unimportant requirements and provides only really necessary information about dependencies between requirements and artifacts.

In this case study the investigator performed the linking step. The investigator started from the user interface of a prototype and tested one requirement after the other. At the same time, he investigated which code pieces were invoked (methods at priority level 1, classes at priority level 2, and packages at priority level 3). He did this investigation by inspecting the code manually. So, the investigator did not perform the linking step concurrently to developing but 'ex post'.

Level 1 linking is the most expensive linking, because the investigator had to link each requirement to every method invoked during its "performance". The investigator developed a traceability matrix containing the requirements as columns and code methods as rows. If, for example, method A participates in the implementation of requirement B, then the cell where the row of method A crosses the column of requirement B contains an 'X'. Each method identifier contains the package name, the class name and the method name, each separated by a period. After the investigator finished level 1 linking, he continued and did the same for precision levels 2 (class level) and 3 (package level).

Results of requirements linking

The effort to create trace links into code at method level is rather high (ca. 45 min per requirement), but, on the other hand, it seems to provide the most useful information with respect to traceability.

Linking into code at class level does not need very much effort (ca. 10 min per requirement). The usefulness of this information depends on class size. For small to medium classes, the level of detail of this information is sufficient to locate the code relevant to a certain requirement. For large classes, e.g., implementing dozens of methods, level 2 linking gives only little support.

Linking into code at package level is done very quickly but does not provide very useful information. Linking at package level therefore is sufficient only for unimportant requirements.

In the case study, requirement linking was performed ex post; therefore it was harder to get an insight into the system. If the linking is done during the project, efforts should be reduced.

The third type of tracing concerns requirements on the lowest precision level 3 (package level). The case study pointed out that tracing at package level can be generally left out, because the resulting traceability matrix provides only very coarse information.

4.5 General Case Study Results

As mentioned above, tracing into code at method level provides more precise information than tracing at class or package level. Unfortunately, the effort for tracing into detailed code is usually very high. The case study pointed out that focusing on the most important requirements reduces efforts in comparison with tracing all requirements at method level in the case study. The total effort for VBRT requirements linking was 770 minutes. That means, it took some 13 hours to establish a VBRT requirements traceability system for a software project with 46 requirements. In comparison, tracing all requirements at method level would have taken 2070 minutes (some 35 hours). Thus, VBRT used around 35% of the effort to establish a full requirements traceability system.

Another interesting point is the usability of VBRT in comparison with ad hoc tracing. The project manager of our case study recognized requirements traceability as additional, time consuming, and expensive effort. In the case study the investigator identified traces ex-post and had problems to get into implementation details, whereas the developers during implementation do not have this problem. So, the case study suggests that capturing traceability information in early phases of the software development lifecycle is much easier than capturing traceability information in later phases. We want to evaluate this hypothesis in further work.

Another issue is the level of detail or precision of tracing. Due to effort and budget constraints it is often impossible to trace each requirement at highest level of detail. Value-based approaches [1, 2] allow tailoring efforts according to requirements' priority. In context with requirements traceability, we interpreted this as to use trace types of variable precision. For example, we used three different trace types to trace the requirements into code, namely method traces, class traces, and package traces. The first trace type allows tracing requirements into code at method level, the second trace type allows tracing requirements into code at class level, and the third trace type allows tracing requirements into code at package level. This reflects a level of precision and also effort necessary to create these traces, e.g., tracing into methods is more expensive than tracing into classes.

It is common knowledge that generally tracing requirements into code at method level provides more detailed and usable information than tracing into class and package level. This is especially true for code that consists of very long source code classes, because the latter often contain methods implementing different parts of functionality. So the information "requirement A is

implemented by methods 1 and 2" is more useful than the information "requirement A is implemented in class X", because there could possibly be many more methods in class X that do not relate to requirement A.

This higher quality of information has its price in effort necessary to create these traces. For instance, the case study presented in this paper pointed out that tracing a requirement into code at method level needed one person for 45 minutes on average to create this trace. Tracing the same requirement into code at class level took only 10 minutes. Tracing the same requirement into code at package level took only 2 minutes, but these traces have very little benefit, because the resulting traceability information is much too coarse, whereas tracing at class level turned out to be sufficient when source code classes are short and clear.

Another question was how "risk" of requirements have an impact on the detail of tracing. Most risky requirements are prone to changes and also need many cycles of adjustment during the process. Thus, it is important to understand the impact of requirement changes on system design and other development artifacts with high precision. This implies that tracing of most risky requirements with high precision has a high benefit, because these traces are needed very often, e.g., during change impact analysis. Furthermore, tracing of risky requirements must be both cheap and fast to allow unobtrusive trace analyzes during software development. Tracing risky requirements also supports the design principle of dividing volatile and less volatile requirements in the design structure.

5. DISCUSSION

The high effort of requirements tracing seems to be a main reason why project teams do not use requirements tracing in practice. Most automation approaches alone do not suffice to tailor down tracing efforts to a manageable size, because they do not reduce complexity of tracing, e.g., the number of traces.

The case study results pointed out that the VBRT approach allows to reduce tracing efforts without losing significant requirements tracing benefits. In comparison to full tracing, VBRT took only 35% effort. Thus, VBRT is a good step towards solving requirements tracing problems like high efforts and high complexity.

Furthermore the case study showed that identification of traces early in the project lifecycle is easier than in later phases. In later phases, e.g., in the operation phase, the rework to get into program details again is considerably higher. This is generally a good argument for requirements tracing, because capturing traces during development is economically much more worthwhile than on the occasion of change requests etc, when the actors do not know implementation details and spend lots of effort to understand the latter.

The prioritization step of the VBRT approach is a suitable means to identify which requirements are more valuable to trace than others. This prioritization is based on requirement parameters like value, risk, and costs and results in reduced efforts, because less important requirements are traced with less efforts and more valuable requirements are traced with more detail. Of course, the prioritization of requirements by the stakeholders is subjective, because it is based on the stakeholder value proposition.

In the case study there were approximately 10% of all requirements in precision/priority level 1, 60% in level 2, and 30% in level 3.

The case study pointed out that all requirements that the stakeholder assessed as high risks (on a scale ranging from high risk, medium risk, to low risk) were assigned to the highest priority level. Thus, we traced them with highest precision. The high risk of these requirements is synonymous with the high volatility of these requirements. That means that requirements with a high probability of change are very valuable to trace.

Based on the case study results, a comparable larger software development project is likely to have the characteristics illustrated in table 4 that depicts strengths and weaknesses of ad hoc tracing, full tracing, and value-based requirements tracing. In the right-most column the optimistic case assumes very little need for extra traces, while the pessimistic case assumes a need for extensive traces to support project activities on the critical path. The overall cost comes from pro-active trace creation and reactive work on tracing when actually needed. We assume the overall tracing-related effort for full tracing in larger projects to be on average approximately 5% of the total project costs as part of quality assurance activities, such as testing or inspection, where traces are a perquisite for a sound quality assurance plan. Ad hoc tracing is likely to cause on average similar but hidden costs, while the cost variation in projects may be very high.

At first sight, *ad hoc tracing* seems to be the cheapest alternative, as there are no costs for trace identification and maintenance. In projects where requirements changes are very likely this alternative might become very costly, because the project team or maintenance personnel have to do "tracing" ad hoc. The later these change requests happen, the more costly tracing gets during development. Further, tracing efforts on activities that are on the critical path for project or maintenance task completion will effectively delay the overall finish. Omitting tracing at that point incurs a high risk of lower-quality solutions and/or erosion of system design [1, 5].

Table 4. Comparison of tracing alternatives

	Proactive effort to identify and maintain traces	Additional effort and delay in case of a change request (reactive)	Overall tracing cost in % of total project costs (optimistic to worst case)
Ad hoc Tracing	Low (0)	Extremely high	0% to 20%
Full Tracing	High	Low	5% to 15%
VBRT	Medium	Low	2% to 7%

The second alternative is *full tracing*. Its effort for trace identification and maintenance is high, because every requirement is traced with the same precision, although many requirements do not need to be traced. Thus, effort is wasted with this variant on many less important requirements, which makes it rather unattractive for practitioners. The general benefit of requirements tracing is the lower delay and lower additional effort in case of a change request.

The third alternative is VBRT. The effort to identify and maintain traces is less than half of full traces and the additional effort in case of change requests is low. Thus, VBRT provides similar value as full tracing, but is much cheaper. Based on our assumptions for a typical project and case study results, the overall tracing effort in a large project with VBRT is likely to be fewer than 3% of the total project costs. This reduction makes tracing advisable in practice, especially after making the hidden costs of ad hoc tracing visible.

Tracing effort depends in practice mostly on the parameters: number of traces, level of detail of traces, change rate of traces at the occasions when tracing is done, e.g., weekly to have a current picture; or, at milestones when artifacts reach a stable state. The number of traces depends on the system artifacts and their size and complexity; the change rate on project context. However, the project manager can control the level of detail of traces and the occasions when tracing gets conducted.

The accurate estimation of tracing effort in general software engineering projects is very difficult as these efforts are often hidden as part of engineering and quality assurance tasks. However, based on a an analysis of the tasks of quality assurance and the amount of effective tracing work involved, we can as initial estimate assume that tracing will consume around a third of quality assurance effort in projects with good quality assurance support. Based on this assumption and typical quality assurance efforts in software development and maintenance projects, the effort estimates in Figure 4 for trace identification during the project and as reaction to change requests seem reasonable. The case study showed that prioritization of requirements by the stakeholders is an effective approach that can lead to a reduction of tracing effort in practice between 30% and 70%. However, the overall savings in a project context depend, of course, on many factors, e.g., how well the system documentation is organized and the overall complexity of the system artifacts. VBRT aims to reduce the inevitable effort for tracing to a level that makes effective tracing more attractive to practitioners.

Validity of results: The purpose of the case study presented in this paper was to investigate the impact of full tracing vs. VBRT on effort and benefits. Thus the case study size was chosen to allow conducting both full tracing and VBRT in a reasonable amount of time. However, the case study project setting is typical in the company and allows reasonable insight into the feasibility of the VBRT process in this environment. We see the empirical investigation as an initial study that supports planning further empirical studies with larger projects. As with any empirical study the external validity of only one study can not be sufficient for general guidelines, but needs careful examination in a range of representative settings.

6. CONCLUSION AND FURTHER WORK

In software development projects there are interdependencies between all kinds of artifacts, e.g. requirements, design, source code, test cases. Requirements tracing is the ability to follow the life of a requirement in a forward and backward direction [8] and helps project managers and project teams to make this interdependencies transparent. Capturing these interdependencies (traces) explicitly brings benefits for identification of requirements conflicts, change impact analysis, release planning etc., but the high complexity and necessary effort of tracing makes requirements tracing too costly for use in practice. Existing methods, tools and approaches of requirements tracing in literature provide only technical models about how to store identified traces and do not take this economic issue into consideration.

In this paper we evaluated the VBRT approach in a real-life project setting and compared costs and benefits of VBRT with ad hoc tracing and full tracing. For the purpose of evaluation, the project team performed the VBRT process steps. We then analyzed the results and compared it with ad hoc tracing and full tracing. Main results of the case study were: (a) VBRT took around 35% effort compared to full tracing; (b) more risky requirements need more detailed tracing. The case study results illustrate that VBRT is an attractive tracing alternative for typical software development projects in comparison with ad hoc tracing and full tracing, because it provides "traditional" benefits of tracing and thereby minimizes tracing efforts.

For a more general evaluation of VBRT and to evaluate the cost difference of VBRT and full tracing in the face of changing requirements we plan multiple case studies with a systematic range of projects. We will address the question which level of detail is optimal to trace requirements into code, e.g. class or method level. Software engineering standards demand requirements traceability but do not state the required level of detail. Further case studies will explore the cost-quality trade-off of tracing at different levels of detail. Automation approaches for requirements tracing are also a future topic in context of VBRT. We want to use the trace analyzer tool [6] to explore the cost-quality trade-offs between automated tracing at method level and class level.

Another focus will lie on improvement of requirements prioritization in order to optimize the value of VBRT. There are many more relevant requirement attributes than value, risk, and effort that are interesting in context with prioritizing requirements for requirements tracing, e.g., architectural relevance, stability. Another open question is how VBRT can support horizontal traceability, because this paper focused on vertical traceability.

A third focus will lie on developing automated support assisting engineers in exploring and using the automatically derived trace dependencies. One idea is to integrate requirements traceability approaches into existing development environments, so that the developer can implement code and store traceability information simultaneously.

Requirements tracing is important to keep track of the interdependencies between requirements and other artifacts and to support project teams and software maintenance personnel in several tasks, e.g. change impact analysis, requirements conflict identification, consistency checking. VBRT is a promising approach to alleviate the problem of high effort of requirements tracing in a practical and comprehensible way.

7. ACKNOWLEDGEMENTS

We want to thank our project partners at Siemens Austria PSE for their time and support during the case study.

8. REFERENCES

[1] B. Boehm, "Value-Based Software Engineering", ACM Software Engineering Notes, 28(2), March 2003

[2] B. Boehm, L.G. Huang, "Value-Based Software Engineering: A Case Study", IEEE Computer, 36(3), 33-41, 2003

[3] J. Bowen, P. O'Grady, L. Smith, "A Constraint Programming Language for Life-cycle Engineering", Artificial Intelligence in Engineering, vol. 5, no. 4, pp. 206-220, 1990

[4] A. Egyed, "A Scenario-Driven Approach to Traceability", Proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001, pp. 123-132

[5] A. Egyed, "A Scenario-Driven Approach to Trace Dependency Analysis", IEEE Transactions on Software Engineering, 2003, pp. 116-132

[6] A. Egyed, P. Grünbacher, „Automating Requirements Traceability: Beyond the Record & Replay Paradigm", Proceedings 17th International Conference on Automated Software Engineering, ASE 2002, pp. 163-171. Edinburgh, IEEE Computer Society

[7] M.W. Evans, "The Software Factory", John Wiley and Sons, 1989

[8] O. C. Z. Gotel, A. C. W. Finkelstein, „An analysis of the requirements traceability problem", 1st International Conference on Requirements Engineering, pp. 94-101, 1994

[9] P. Grünbacher, A. Egyed, N. Medvidovic, "Reconciling Software Requirements and Architectures with Intermediate Models", Software and System Modeling (SoSyM), Vol. 3, no. 3, Springer, pp. 235-253, 2004, ISSN: 1619-1366

[10] S.D.P. Harker, K.D. Eason, "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering", IEEE, 1992

[11] P. Hsia, J. Gao, J. Samuel, D. Kung, Y. Toyoshima, C. Chen, „Behavior-based Acceptance Testing of Software Systems: A Formal Scenario Approach", IEEE, 1994

[12] J. Jackson, "A Keyphrase Based Traceability Scheme", IEE Colloquium on Tools and Techniques for Maintaining Traceability during Design, 1991, pp.2-1-2/4

[13] H. Kaindl, "The Missing Link in Requirements Engineering", ACM SigSoft Software Engineering Notes, vol. 18, no. 2, pp. 30-39, 1993

[14] J. Karlsson, "Software Requirements Prioritizing", Proceedings of the 2nd International Conference on Requirements Engineering (ICRE'96), Colorado Springs, Colorado, April 15-18, 1996

[15] M. Lefering, "An Incremental Integration Tool between Requirements Engineering and Programming in the Large", Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, California, Jan. 4-6, pp. 82-89, 1993

[16] A. Ngo-The, G. Ruhe, "Requirements Negotiation under Incompleteness and Uncertainty", Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay (SEKE '03), July, pp. 586-593, 2003

[17] M.C. Paulk, B. Curtis, M.B. Chrissis, C.V. Weber, "Capability Maturity Model for Software", Version 1.1, Technical Report, CMU-SEI-93-TR-024, February 1993

[18] F.A.C. Pinheiro, J. A.Goguen, "An Object-Oriented Tool for Tracing Requirements". IEEE Software 13(2), 1996, 52-64.

[19] B. Ramesh, T.Powers, C. Stubbs, M. Edwards, "Implementing Requirements Traceability: A Case Study", IEEE, 1995

[20] B. Ramesh, M. Jarke, "Toward Reference Models for Requirements Traceability", IEEE Transactions on Software Engineering, Vol. 27, No. 1, pp. 58-93, January 2001

[21] G. Ruhe, D. Greer, "Quantitative Studies in Software Release Planning under Risk and Resource Constraints", International Symposium on Empirical Software Engineering (ISESE '03), pp. 262-271

[22] R. Watkins, M. Neal, "Why and how of Requirements Tracing", IEEE Software, vol. 11, no. 7, pp. 104-106, July 1994