Delft University of Technology
Software Engineering Research Group
Technical Report Series

# Monitoring Requirements Coverage Using Reconstructed Views: An Industrial Case Study

Marco Lormans, Hans-Gerhard Gross, Arie van Deursen,
Rini van Solingen and André Stehouwer

Report TUD-SERG-2006-014

TUD-SERG-2006-014

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
`http://www.se.ewi.tudelft.nl/techreports/`

For more information about the Software Engineering Research Group:
`http://www.se.ewi.tudelft.nl/`

Note: Accepted for publication in the Proceedings of the Working Conference on Reverse Engineering (WCRE), 2006, IEEE Computer Society.

# Monitoring Requirements Coverage using Reconstructed Views: An Industrial Case Study

Marco Lormans and Hans-Gerhard Gross
*Delft University of Technology*
*(M.Lormans, H.G.Gross)@ewi.tudelft.nl*

Arie van Deursen
*Delft University of Technology and CWI*
*Arie.vanDeursen@tudelft.nl*

Rini van Solingen
*LogicaCMG and Drenthe University*
*Rini.van.Solingen@logicacmg.com*

André Stehouwer
*LogicaCMG*
*Andre.Stehouwer@logicacmg.com*

## Abstract

*Requirements views, such as coverage and status views, are an important asset for monitoring and managing software development. We have developed a method that automates the process for reconstructing these views, and built a tool, ReqAnalyst, to support this method. In this paper, we investigate to what extent we can automatically generate requirements views to monitor requirements in test categories and test cases. The technique used for retrieving the necessary data is an information retrieval technique called Latent Semantic Indexing (LSI). We applied our method in a case study at LogicaCMG. We defined a number of requirements views and experimented with different reconstruction settings to generate these views.*

## 1. Introduction

A "requirements view" on a system or development process offers a perspective on that system in which requirements are leading [18]. For example, requirements views can describe project progress in terms of testing (these requirements have been successfully tested), design (the requirements that resulted in a design decision), coding (the requirements that were actually implemented), and so on.

Requirements views are essential for successful project management, in order to monitor progress in product development. In an outsourcing context, reporting progress in terms of requirements is particularly important, since the customer is much less aware of the system breakdown or implementation issues, and is primarily interested in his requirements.

Unfortunately, capturing, monitoring, and resolving multiple views on requirements is difficult, time-consuming as well as error-prone when done by hand [17]. The creation of requirements views requires an accurate traceability matrix, which in practice turns out to be very hard to obtain and maintain [6, 10, 13, 21].

To remedy this problem, a significant amount of research has been conducted in the area of reverse engineering traceability links from available software development work products [8, 14, 19]. Our own line of research has focused on the use of information retrieval techniques, in particular *latent semantic indexing* (LSI) [3], for this purpose, and the application of the reconstructed matrices for view reconstruction specifically [11, 12].

While significant progress in these areas has been booked, a number of open research issues exist, which we seek to explore in this paper.

The first question we address is which requirements views are most needed in practice. To answer this question, we have sent out a questionnaire to a dozen practitioners. From the answers, we have distilled three important groups of views, which are described.

The second question is how and to what extent these particular requirements views can be reverse engineered from existing work products. Can the approach we proposed in [11, 12] be used to reconstruct these views? Our answer comes in the form of a prototype tool, called ReqAnalyst, which implements a way of reconstructing these views, offering project stakeholders the capabilities to inspect the system and development progress in terms of these views.

The third and hardest question is if these reconstructed views can help in a real life software development process. To address this issue, we take an extensive look at a long running, complex software development process that has been going on for several years.

The project at hand deals with a traffic monitoring system (TMS). The development of TMS is outsourced to LogicaCMG, an international IT services supplier. Progress reporting to the customer must be done in terms of requirements, making accurate requirements views an essential success factor in the project. We discuss the way of working in this project, and analyze to what extent reconstructed links can be used to support and enhance the way of working. In our case study we focus on requirements views related to testing.

The remainder of this paper is organized as follows. In Section 2 we discuss existing work in the area of requirements views and reverse engineering of traceability matrices. In Section 3, we summarize our methodology for generating requirements views, based on [11, 12]. In Sections 4, 5, and 6 we present the requirements views we aim at, the ReqAnalyst tool, and the case at LogicaCMG, respectively, after which we conclude the paper with a summary of contributions, and suggestions for future research.

## 2. Related Work

The term 'view' is often used in the area of software engineering, especially in the area of requirements engineering. Views are generally introduced as a means for separation of concerns [18] and mostly represent a specific perspective on a system. Nuseibeh *et al.* discuss the relationships between multiple views of a requirements specification [18]. Most systems that are developed by multiple participants have to deal with requirements that overlap, complement and contradict each other. Their approach focuses on identifying inconsistencies and managing inconsistencies in the requirements specification.

Another approach is to use a well structured document set, conforming to known templates such as MIL-std 498, Volare or IEEE-std-830-1998. These templates help in getting an overview of what the system does, but they are often not sufficient. Project managers, but also other team members, need fast access to this data, and, preferably, they would like only a subset of the whole pile of documents produced during the development life-cycle. Current templates are not sufficiently flexible and are difficult to keep consistent during development.

Nissen *et al.* show that meta-models help managing different requirements perspectives [17]. The meta-models define what information is available and how it is structured in the life-cycle: the development artifacts including their attributes, and the traceability relations that are allowed to be set between these artifacts. An important area of research is developing these meta-models [16, 20, 22, 23, 25], which constrain the views that can be generated.

Von Knethen proposes traceability models for managing changes on embedded systems [23, 24]. These models help estimating the impact of a change on the system, or help to determine the links necessary for correct reuse of requirements. According to Von Knethen, defining a workable traceability model is a neglected activity in many approaches. Our earlier research confirms the importance of defining a traceability model [13]. Some initial experiments concerned a static traceability model. New insights suggest a dynamic model, in which new types of links can be added as the way of working evolves during the project. The need for information as well as the level of detail changes constantly for big development projects [6].

In order to reconstruct requirements views from project documentation we need traceability support. Several traceability recovery methods and supporting tools already exist, each covering different traceability issues.

De Lucia *et al.* present an artifact management system, which has been extended with traceability recovery features [14]. This system manages different artifacts produced during development such as requirements, designs, test cases, and source code modules. De Lucia *et al.* use latent semantic indexing (LSI) for recovering the traceability links. They also propose an incremental traceability recovery process in which they incrementally try to identify the 'optimal' threshold [15].

Natt och Dag *et al.* [19] and Huffman Hayes *et al.* [8] primarily use traceability reconstruction for managing requirements of different levels of abstraction, such as between business requirements and system requirements. Natt och Dag *et al.* discuss their approach and tool, ReqSimile, where they have implemented the basic vector space model and applied it in an industrial case study [19]. Huffman Hayes *et al.* have implemented various methods for recovering the traceability links in their tool called RETRO [8]. They also applied their approach in an industrial case study.

Cleland-Huang *et al.* define three strategies for improving dynamic requirements traceability performance: hierarchical modeling, logical clustering of artifacts and semi-automated pruning of the probabilistic network [1].

## 3. A Methodology for Generating Requirements Views

In our earlier work [11], we have proposed an approach for reconstructing requirements views and we experimented with the reconstruction of traceability links in several case studies [12]. Our method consists of the following six steps:

**Step 1: Defining the traceability meta-model.** The underlying traceability meta-model defines the work products and the type of links that are permitted. Examples can be found in [16, 20, 22–25].

**Step 2: Identifying the work products.** The work products are identified in the provided documentation. Each work product is given an unique identifier, for example, 'FR01' for a requirements description.

**Step 3: Preprocessing the work products.** Each work product is preprocessed to support automated analysis. The text of each work product needs to be extracted and transformed into plain text. This includes typical information retrieval steps such as lexical analysis, and so on.

**Step 4: Reconstructing the traceability links.** The traceability links are reconstructed for which we use Latent Semantic Indexing [3]. The result of this step is the complete set of candidate traceability links.

**Step 5: Selecting the relevant links.** The possible relevant links are automatically selected from the complete set of candidate links using various link selection strategies.

**Step 6: Generating requirements views.** Finally, the requirements views are generated using the reconstructed traceability links.

## 4. Which Views are Needed in Practice?

To determine which requirements views are needed in practice, we set up a questionnaire and distributed it among various practitioners. Below we describe the process we used for this, as well as the three main types of views that emerged from our investigation.

### 4.1. Requirements View Questionnaire

The goal of our questionnaire is to get an impression which views are helpful and what information these views should present. We distributed the questionnaire among people holding various roles within the software development life-cycle. The roles we distinguished are: project manager, software process improvement / quality manager, product marketing manager, requirements engineer, system/software architect, programmer and test engineer, as well as more specific roles such as product owner and usability designer.

The respondents came from the industrial partners of the Merlin[1] project we are involved in. This is a European research project in the area of global software development in which various universities and companies participate. In total, the questionnaire was spread among all 7 industrial partners. We got a response from 5 of the companies with multiple filled in questionnaires. In total we had 12 fully filled in questionnaires containing around 100 descriptions of desirable views for different roles in the life-cycle.

We also asked if these views can be extracted from the work products they currently produce during the development life-cycle. Most respondents think that this is possible, because this information should be stored somewhere in the work products. However, the exact location of this information is not always known.

We have learned from this questionnaire that the possibility for browsing requirements data and the underlying work products is essential in all environments. A challenge here is that in many cases the readability of many of the work products leaves much to be desired, and that it is often hard to get an overview of the whole system. In addition to that, stakeholders can easily get lost when looking for information if there are too many possible links to follow. Our views should take care of this issue, and should make it easier to arrive at the exact information one needs for the view in question.

Furthermore, we learned from this questionnaire that the following information is desirable in a requirements view:

- For each requirement their source, description, motivation, importance, history, status and dependencies to other work products.
- For each group of requirements a list of all requirements, the status of their implementation and verification (not tested, test passed, test failed).
- Life-cycle paths; per requirement the complete path it undergoes during the life-cycle. In other words, walking the complete path of dependencies per requirement (using traceability). Two paths are of interest for the developers: the Requirements-Implementation path and the Requirements-Test path.
- For all the requirements the coverage in a certain work product. These work products can, for example, be a lower level of requirements, the design or the test cases.

From the questionnaire we can conclude that various developers and managers are interested in specific information about a certain requirement (see first and third bullet) or a group of requirements, sometimes in relation to other work products (see last bullet).

From the answers to this questionnaire we distilled three types of views: Coverage views, Life-cycle Path views, and Status views, which we will briefly discuss below.

### 4.2. Coverage Views

Requirements coverage views focus on the localization of the requirements in the rest of the system. These views show if and where a certain requirement is covered in the system. This can be coverage in, for example, the system architecture, in the detailed design, or in the test cases. The number of different types of coverage views depends on the

[1]www.merlinproject.org

meta-model defined for the development process. It prescribes which phases are defined and what work products are produced during these phases.

According to Costello *et al.* requirements coverage is defined as: *The number of requirements that trace consistently to the next level up or down* [2]. Costello *et al.* originally defined this metric for requirement to requirement coverage. As this definition is very general, it is also suitable for the coverage of requirements to other work products.

Hull *et al.* also define three so called traceability metrics [9]. One of them, *Traceability Breadth*, relates to coverage. It measures the extent to which requirements are covered by the adjacent layer above or below (within the defined meta-model).

We define requirements coverage as follows: If a link between a requirement and another work product, e.g. a test case, exists and this link is correct, the requirement is covered by that work product. In the requirements coverage view we show which requirements are covered by work products as well as the percentage of these requirements with respect to the total number of requirements. For example, we can define the percentage of requirements that are covered by a test case as follows:

$$coverage_{test} = \frac{|requirements_{test}|}{|requirements_{total}|},$$

where $coverage_{test}$ represents the coverage in the test case specification, $requirements_{test}$ the number of requirements traced consistently by test cases and $requirements_{total}$ the total number of requirements.

This coverage metric is very general and can be used for requirements coverage in other life-cycle phases as well, such as the coverage of requirements in the design.

### 4.3. Life-cycle Path Views

From the questionnaire we learned that two life-cycle paths are important: the Requirements-Implementation path and the Requirements-Test path. When comparing this to the well-known V-model, we see that these are the horizontal and vertical dimensions of this life-cycle model.

The second traceability metric Hull *et al.* defined, *Traceability Depth*, is useful for this view [9]. This metric relates to the number of layers the traceability extends. These layers are captured in the life-cycle path.

The last traceability metric discussed by Hull *et al.* is also interesting with respect to our life-cycle path views [9]. This metric, *Traceability Growth*, measures how a requirement expands down through the layers of the meta-model (in our case the life-cycle path). For example, a requirement can be covered by one test case or by multiple test cases. For impact analysis this is a useful metric to include in our life-cycle path view.

### 4.4. Status Views

Status views concern the status of a (set of) work product(s) such as a (set of) requirement(s). The view shows a specific status of the work product in the life-cycle. For example, given the presence of a link, the status of a requirement can be appropriately set; the requirement is dealt with in the other work product. Moreover, management information can be obtained by computing percentages of requirements that have reached a certain status.

Often traceability support is not enough to generate complete status reports of requirements, for example, when a project manager needs to know if all requirements have passed a test. Traceability can help identifying the requirements in the test document (the document that describes the test), and hopefully also in the test report document. The latter contains the information if a requirement has passed the test. This information needs to be extracted from the document and included in the status view as well.

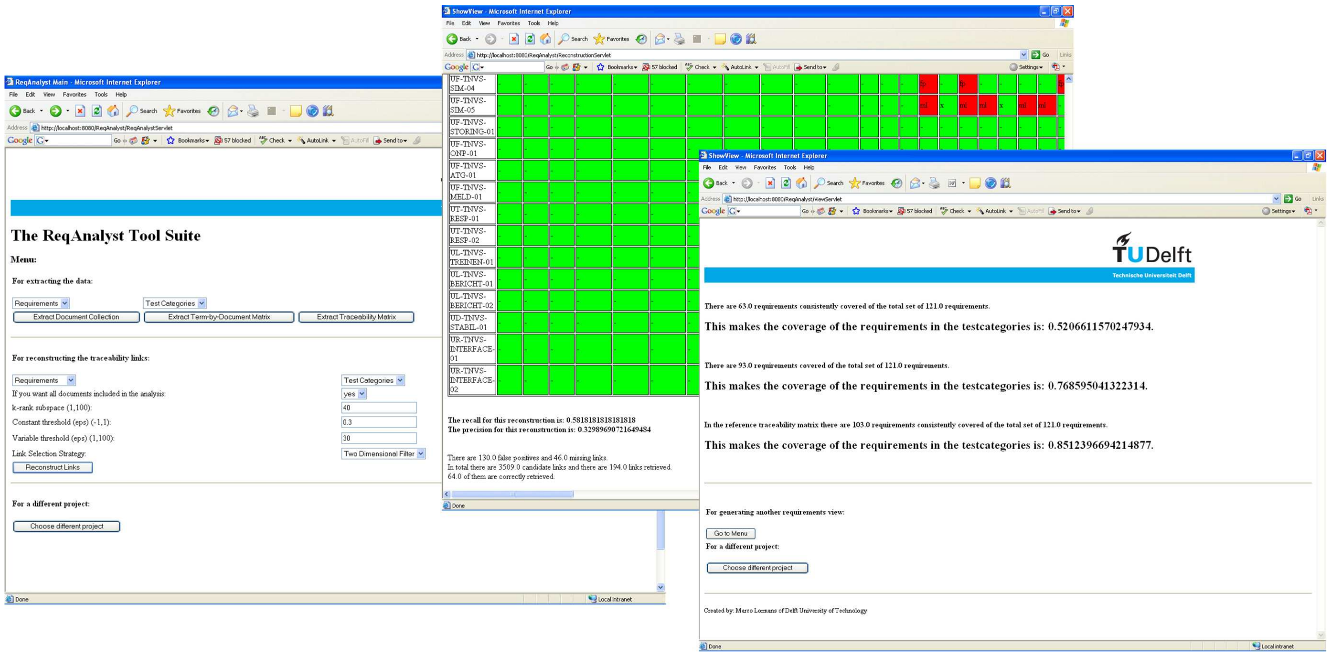
In our case study we like to monitor this extra status information and not only the traceability data. We would like to retrieve ’richer information’ concerning the status of the requirements. For example, a status view for an individual requirement can show its relations to other work products (coverage) including its status such as ’covered by test, but not tested yet’, ’covered by test, and failed the test’ or ’covered by design, but not covered by test’.

All three views should make it possible to obtain continuous feedback on the progress, in terms of requirements, of ongoing software development or maintenance projects. Furthermore, they facilitate communication between project stakeholders and different document owners.

## 5. The ReqAnalyst Tool Suite

For supporting our approach we developed a tool called ReqAnalyst. For this tool suite, we used the Extract-Query-View approach [5]. In this approach, we first extract the relevant data from the provided documents. This data, the work products and if available the reference traceability matrices, is stored in a database. The reference traceability matrix is the matrix that contains the correct links according to the experts. For reconstructing the traceability links, queries can be done on the database. The reconstructed information combined with the data from the database is used to generate the requirements views.

ReqAnalyst is implemented using standard web-technology. For storing the data we use a MySQL database. On top of the database we have implemented a Java web application using Java Servlets and Java Server Pages (JSP). The choice for building a dynamic web application in Java made it easy to fulfill a number of the practical tool require-


The ReqAnalyst Tool Suite

Menu:

For extracting the data:

Requirements | Test Categories

Extract Document Collection | Extract Term-by-Document Matrix | Extract Traceability Matrix

For reconstructing the traceability links:

Requirements | Test Categories

If you want all documents included in the analysis: yes

k-rank subspace (1,100): 40

Constant threshold (eps) (-1,1): 0.3

Variable threshold (eps) (1,100): 30

Link Selection Strategy: Two Dimensional Filter

Reconstruct Links

For a different project:

Choose different project

The recall for this reconstruction is: 0.5818181818181818
The precision for this reconstruction is: 0.32989690721649484

There are 130.0 false positives and 46.0 missing links.
In total there are 3509.0 candidate links and there are 194.0 links retrieved.
64.0 of them are correctly retrieved.

There are 63.0 requirements consistently covered of the total set of 121.0 requirements.

This makes the coverage of the requirements in the testcategories is: 0.5206611570247934.

There are 93.0 requirements covered of the total set of 121.0 requirements.

This makes the coverage of the requirements in the testcategories is: 0.768595041322314.

In the reference traceability matrix there are 103.0 requirements consistently covered of the total set of 121.0 requirements.

This makes the coverage of the requirements in the testcategories is: 0.8512396694214877.

For generating another requirements view:

Go to Menu

For a different project:

Choose different project

Created by: Marco Lormans of Delft University of Technology

Figure 1. An example ReqAnalyst session

ments such as ease of deployment[2]. Furthermore, every project member can use a browser to access the tool independent of his or her location, making it suitable for global distributed software development.

## 5.1. Features of ReqAnalyst

The functionality of the present version of ReqAnalyst is still relatively simple. ReqAnalyst currently is primarily a research prototype, allowing us to experiment with the use of LSI for requirements view reconstruction.

A ReqAnalyst session starts by choosing a project, which can be a new one, or one that has been stored in the database already. Once the user has chosen a project, ReqAnalyst shows a menu with the steps that can be executed next. This main menu follows the steps from the Extract-Query-View approach [5]. The first submenu provides functionality for extracting the data from the provided documentation. The work products and the reference traceability matrices can be extracted. The second submenu provides the options for setting the parameters of the LSI reconstruction and the choice for a link selection strategy. Finally, the bottom menu provides an option for choosing a different project.

Once the tool has executed a reconstruction, an intermediate menu appears showing the reconstructed traceability matrix and some options for generating various requirements views. This intermediate menu shows all the metrics relevant for assessing the reconstruction, such as recall, precision and the number of false positives and missing links. In addition to that, ReqAnalyst offers views that support the comparison of traceability matrices obtained in different ways, e.g. manual versus automatically via LSI.

In Figure 1 a session with ReqAnalyst is shown, in which the tool is used to analyze a set of requirements. Concrete contents of requirements have been made anonymous in order to protect the customer's interest. The leftmost window displays the main menu for setting the parameters and starting the analysis. The window in the middle shows the results of a reconstruction including the metrics for assessing the results. The rightmost window shows an example of a coverage view. The view compares the coverage of correct retrieved links with all retrieved links (including false positives) and the coverage of the provided reference traceability matrix. The list of related requirements is not shown. Note that all interactions take place via a standard browser.

An important feature of ReqAnalyst is the possibility to browse the reconstructed results. It allows users to inspect the reconstructed traceability matrix and browse the traceability links (implemented as hyperlinks). Furthermore, the reconstructed matrix can be compared to a reference matrix, if available. The correct links, colored green, and the incorrect links, colored red, can be analyzed. When follow-

[2]For our case study we used the Apache Tomcat 5.5 web server for deployment

ing the hyperlink, all the information concerning the two entities involved can be browsed and inspected.

### 5.2. Views in ReqAnalyst

**Coverage Views.** The 'Coverage View' as implemented in ReqAnalyst shows the number of requirements that are correctly covered in the other work product and the total number of requirements that are analyzed. It also shows the coverage percentage as defined in Section 4.2. Finally, it lists the requirements with their description and the related artifacts of the other work product. Besides the coverage, it is also possible to see which requirements are not covered by the other work product.

**Life-cycle Views.** ReqAnalyst supports the reconstruction of links between two work products. These two work products can be the beginning of a life-cycle path and the end of a life-cycle path. ReqAnalyst reconstructs the traceability between these concepts considering it as one link in the meta-model. Currently, ReqAnalyst can not automatically derive the traceability matrices that cross multiple layers in the meta-model. In other words, it ignores the traceability data between intermediate concepts. However, in this view it should take into account this intermediate data to show the complete path between the two work products. ReqAnalyst is not able to combine the data from intermediate layers in a single view.

**Status Views.** The 'Status View' is not yet implemented in our ReqAnalyst tool. Currently, we only extract the relevant data to reconstruct our traceability links. For the status views, additional status attributes need to be extracted from the provided documentation, which is left as future work.

## 6. Case Study: LogicaCMG

For many companies traceability support is a major challenge [7, 8, 13]. The problem can be best explained according to a real life example. In this section we will discuss the way of working at LogicaCMG in monitoring the progress of requirements as well as the results from an industrial case study we conducted at LogicaCMG. First, we discuss some experiences at LogicaCMG and how a project handles progress monitoring of requirements. Next, we describe how we applied our methodology as a pilot project parallel to this project.

### 6.1. Case Background

The project in our case study involves a traffic monitoring system (TMS), which is an important part of a traffic control and logistics system. The main purpose of TMS is to record the positions of vehicles in the traffic system. These recordings are used to adjust the schedules of running and planned vehicles as well as operating the necessary signaling.

**Initial Approach** In our earlier work, we discussed the setting that LogicaCMG initially used in the TMS project for handling requirements management in this outsourcing context [13]. Below, we summarize this approach, and discuss its shortcomings.

In the initial TMS setting, LogicaCMG used IBM Rational RequisitePro for managing the requirements and MIL-std-498 [4] for documenting their work products. The project consumed 21 man years in the last 3 years of development. In total, there are over 1200 requirements and over 700 test cases. All the traceability links between the work products needed to be manually set. This manual work, which is time-consuming and error-prone, is acceptable if it is a one time task. However, when requirements change or new requirements come in, the links can become inconsistent; old links may need to be dropped and new links may need to be added. Sometimes the huge number of changes caused that the effort needed for updating the traceability links was comparable with resetting all the links.

An additional issue in this setting is the fact that the customer was not willing to operate on the tagged documentation LogicaCMG provided along with the tool, since they wanted to maintain their own documents. For managing the requirements in this particular case, LogicaCMG was forced to make separate requirements documents in which the traceability was manually set by the requirements engineers. The main shortcomings of this setting are:

- Unreliability, as the consistency of the traceability links could not be guaranteed. It was hard to keep the links consistent during the evolution of the project.
- The manual work for synchronizing the updates from the client introduced errors, was time consuming and cost the project an unbalanced effort.

This makes the information for monitoring the progress of the requirements during the development process unreliable. It increases risks during the integration phase, such as requirements that are not implemented or functionality that should not be implemented in the system.

Currently, an alternative way of working is introduced at LogicaCMG to overcome these shortcomings. The initial setting did not allow simple improvements to tailor the setting according to their development needs.

**The Current Setting.** Compared to the way of working described in the previous paragraph, LogicaCMG has set up an alternative setting for further developing the TMS system. All work products are still documented according to MIL-std-498 [4], but currently they are all maintained by LogicaCMG. This includes the Requirements Specifications and the Software Test Descriptions. Synchronizing changes is easier now as the development methodology for all documents is equal.

Instead of managing all traceability links, only the essential links are managed. The number of possible link types for testing are reduced, making the meta-model less complex. As a result, the reduction of possible traceability links also reduces the risk of inconsistencies.

This reduction of links was realized by merging the test scripts and the test documentation. So, the new test cases, written in 'tst'-files, now include the description of the test as well as the real script for executing the test. These 'tst'-files also include the unique identifier of the requirements they cover. Currently, these identifiers are manually set.

The explicitly documented requirement identifiers are actually the traceability links between the requirements and test cases. The test cases are structured (with the requirements identifiers) so that Doxygen [3] can generate a HTML representation of the test cases including hyperlinks to the requirements.

In this way of working the traceability links are still manually set. Our approach aims at automating this. The case study at hand offers us an opportunity to show that our approach can be useful in practice and that it can reduce the effort needed for consistent traceability support.

### 6.2. Case Configuration

In the TMS case study, we investigate the relation between requirements and test categories and between requirements and test cases. More specifically, we focus on the requirements-to-test-coverage and the requirements-test-path views.

Two main documents are provided: a System/Subsystem Specification (SSS), containing the requirements and a Software Test Description (STD), containing the description of the test categories. Both are MS-Word documents and are structured according to MIL-std-498 [4]. This means that traceability data is incorporated in these documents and that it is possible to extract a reference traceability matrix from this data.

Besides the two MS-Word documents, a HTML document generated by Doxygen is provided. This document is an addition to the STD and contains the description of the test cases. It also contains the description of the test categories and, in some cases, of the requirement(s) it refers to (see Section 6.1). The HTML document is accompanied by a MS-Excel spreadsheet, which contains the traceability links between the requirements and the test cases.

[3]`www.doxygen.org`

| | |
|---|---|
| Number of Requirements Categories | 43 artifacts |
| Size of Requirements Categories | 1168 terms |
| Average number of terms per document | 183 terms |
| Number of Requirements | 121 artifacts |
| Size of Requirements Documents | 695 terms |
| Average number of terms per document | 29 terms |
| Number of Test Categories | 29 artifacts |
| Size of Test Categories | 589 terms |
| Average number of terms per document | 183 terms |
| Number of Test Cases | 98 artifacts |
| Size of Test Cases | 886 terms |
| Average number of terms per document | 107 terms |
| Total number of indexed terms | 1783 terms |
| Average number of terms per document | 93 terms |

Table 1. TMS Case Study Statistics

Our meta-model for this case study is shown in Figure 2. In this model we can identify the following work products. First of all, the scenarios describe a general use case. Since the scenarios were documented in a memo and as this is not a formal document in MIL-std-498, we ignored the scenarios in this case study. Furthermore, in the SSS a hierarchy can be identified. The uniquely identifiable requirements are clustered according to a hierarchy resulting in categories of requirements. Just like the individual requirements, these requirements categories have a unique numbering, which is why we also took these into account for our analysis as well.

Examples of requirements categories are general categories such as goal and domain, but also more specific ones such as the use of computer resources, specific system interfaces and safety. Each of these requirements categories has one or more uniquely identifiable requirements. Naturally, the traceability between the requirements categories and requirements can be derived from the hierarchy. This traceability is not explicitly incorporated in the MS-Word documents. The SSS does contain the traceability links between the scenarios and the individual requirements.

For the test cases we can identify the same hierarchy resulting in the separate work products “test category” and “test case”. Both are uniquely identifiable in the provided documentation. The STD contains the traceability links between the requirements and the test categories.

The bold lines in Figure 2 are the links that LogicaCMG currently maintains (in the SSS, STD and MS-Excel). The other lines are the links that can be derived indirectly by the hierarchical structure of the documents.

In total, we monitored the progress of 121 requirements

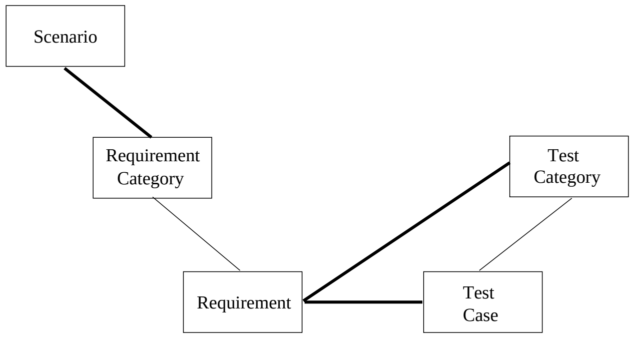


Figure 2. Traceability Meta-Model

in this case study. As these requirements are provided by MS-Word documents we needed to do some manual processing to extract the relevant data from the SSS and store the processed tokens of text in our database. The requirements have an unique identifier and consist of a description. Besides the requirements, the document contains some context explaining certain domain knowledge for a group of requirements. We extracted this data as well and stored it in our database and marked it as "context".

For the test categories and test cases, the same approach for obtaining the relevant data can be used resulting in 29 test categories and 98 test cases.

## 6.3. Results

**Quality of Reference Matrix.** We expect our methodology should reduce the effort for maintaining consistent traceability. This means that the effort for getting a validated traceability matrix is less than doing it manually. For this, we first need to check the quality of our reference traceability matrix.

The initial results are generated using ReqAnalyst and the reference traceability data provided by the experts of LogicaCMG. Next, we conducted one validation session and reconstructed the links again with the updated information. We executed this session as it is hard to maintain a consistent traceability matrix by hand. So, it is unlikely to assume that the reference traceability data for our TMS case study is consistent.

We have used the 20% variable threshold as input for our validation session. For this session it took the expert about 30 minutes to inspect the 31 false positives and 61 missing links. It resulted in resetting 4 missing links. These links initially were indicated as link by the expert, but because ReqAnalyst did not reconstruct them, the expert reassessed the links and decided to remove them from the reference traceability data. This improved traceability data is used as reference in our reconstruction.

**Reconstruction Settings.** Our reconstruction based on LSI can be tuned in several ways. In Table 2, we show the reconstruction results of the requirements to test categories. The reconstruction between the other work products shows equal results. For the meaning of the parameters we refer to [12]. We have found the best results with a reduced rank-k subspace of 40% and a constant threshold of $c$ = 0.3. We varied with the parameter $\varepsilon$, which indicates the variable threshold for our link selection strategy. The 20% in Table 2 means that only 20% of the candidate links are selected as traceability links.

In total for this reconstruction there are 3509 candidate links and 110 links in the reference traceability matrix (the correct positives plus missing links is always 110).

**Reconstruction Quality.** The recall (correct positives / total reference links) and precision (correct positives / total reconstructed links) show expected results for an industrial case study [12, 15].

For our application, the results of the last two columns, percentage of validation work and coverage percentage, are the most interesting. The percentage of validation work refers to the effort needed to validate the reconstructed links manually compared to validating all possible candidate links manually (total reconstructed links / total candidate links). The coverage percentage refers to the percentage of correctly covered requirements compared to all the requirements (see Section 4.2).

A validation percentage of 2% means that the developers only need to validate 2% of all the candidate links manually. A low validation percentage is positive as it indicates the effort needed to keep the traceability support consistent after e.g. a change. In this example, 98% of the candidate links do not need to be validated again.

However, in the case where the validation percentage is 2%, there are also correct links missing compared to the reference traceability matrix, namely 57 missing links. This is not acceptable. In practice, the goal is to achieve 100% recall, so only false positives need to be eliminated. Table 2 shows that with a constant threshold of $c$ = 0.3 we never achieve a recall of 100%. So we decreased $c$ to 0.2 and 0.1. With $c$ = 0.1 we reached a recall of almost 100%. Unfortunately, the number of false positives increases and accordingly the validation percentage. Still, the total effort reduction is 35%. From these results we can conclude that it is very hard to recover the last 10–15 missing links with our approach and realize a recall of 100%. As such it makes sense to investigate which textual revisions are needed in the documents that would enable automatic recovery.

The second interesting column, the coverage percentage, increases as the recall increases. This is expected behavior as it uses the correct positives as input and ignores the false positives. As the recall approaches 100%, the cov-

| Link Type | $\varepsilon$ | Reconstructed Links | | Missing Links | Recall | Precision | Validation Percentage | Coverage Percentage |
|---|---|---|---|---|---|---|---|---|
| | | Correct Positives | False Positives | | | | | |
| Requirements | 20% | 53 | 31 | 57 | 0.48 | 0.63 | 2 | 43 |
| to | 40% | 76 | 329 | 34 | 0.69 | 0.19 | 12 | 62 |
| Test | 60% | 83 | 728 | 27 | 0.75 | 0.10 | 23 | 68 |
| Categories | 80% | 83 | 747 | 27 | 0.75 | 0.10 | 24 | 68 |
| $c = 0.2$ | 80% | 95 | 1392 | 15 | 0.86 | 0.07 | 42 | 77 |
| $c = 0.1$ | 80% | 107 | 2159 | 3 | 0.97 | 0.05 | 65 | 83 |

Table 2. Results Two Dimensional Filter Strategy on TMS case with rank-k subspace of 40% and $c = 0.3$

erage percentage will get closer to the coverage that is obtained from the reference matrix, which presently is 85%. In TMS case study, currently, 85% of the requirements are covered by test categories.

## 6.4. Lessons Learned

**Consistent Traceability Support.** The first observation is the fact that we found some small inconsistencies during our analysis. The traceability data incorporated in the SSS and the traceability data maintained in MS-Excel show different links compared to the content of the descriptions. For example, a requirement that was cancelled, was still included in the traceability data. The manual synchronization of these work products is apparently error-prone. ReqAnalyst can identify these inconsistencies, after which the developer can correct it. This way, maintaining consistent traceability support becomes easier.

**Effort Reduction.** It is harder to estimate if ReqAnalyst really reduces the effort needed for keeping the traceability support consistent. Is the 35% effort reduction reasonable? In our case, we did a first-time reconstruction and one increment (the validation session). Following increments can take into account the validated reference traceability matrix. So, false positives that are already discarded from a previous reconstruction are ignored. We expect that this will again reduce the effort for doing a next update. However, ReqAnalyst does not support this automatic validation yet. We updated our reference traceability data manually after the validation session with the expert.

**Requirements Views.** Although ReqAnalyst does not support all defined views yet, it increases developers' insights in the system. Our views improve the possibilities to systematically review and validate the requirements. Individual requirements can be inspected with respect to their coverage and their role within the system.

An issue is the fact that our views greatly depend on ReqAnalyst's traceability support (as discussed above). Once the traceability is consistent, the progress of requirements can easily be monitored with the defined requirements views.

**Quality of the Documentation.** Our validation session also improved the quality of the content of the work products. Normally, the specifications are reviewed by individual persons after a change. In our validation session, we inspected the false positives and missing links. Assessing the links, implied reviewing the descriptions of the related work products. This also led to more harmonized descriptions in the documentation. It is worth investigating what the documentation requirements are to enable full automated traceability with a 100% recall. If projects could improve their documentation and that would enable fully automated traceability reconstruction, the benefits for practice would increase considerably.

## 7. Conclusions

In this paper, we have studied the reverse engineering of requirements views from software development work products, in the context of an industrial outsourcing project. We consider the following as our key contributions:

- We identified, through a questionnaire among practioners, what relevant requirements views are;
- We demonstrated how these requirements views can be reconstructed, and implemented this reconstruction in our ReqAnalyst tool suite;
- We applied our approach to an ongoing project at LogicaCMG.

Our future work will concern the following issues. First, we would like to tune our approach and come to more specific guidelines to reduce the effort needed to get a validated reference traceability matrix. Furthermore, we would like to expand the number of requirements views for more complex environments with more sophisticated meta-models and were we can generate 'richer' requirements views such as our life-cycle and status views. Last but not least, we are starting up a new industrial case in the area of consumer electronics. This case concerns a global distributed software development environment and a product-line, making it a very complex environment to apply our methodology.

**Acknowledgments** We would like to thank the Merlin partners for filling in the questionnaire. In particular we would like to thank LogicaCMG and the members of the TMS project for their cooperation and making this research possible. Partial support was obtained from NWO Jacquard, project Reconstructor, and SenterNovem, project Single Page Computer Interaction (SPCI).

## References

[1] Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proc. of the 13th IEEE Int. Conf. on Requirements Engineering*, pages 135–144, Washington, DC, USA, 2005. IEEE Computer Society.

[2] Rita J. Costello and Dar-Biau Liu. Metrics for requirements engineering. *Journal of Sys. and Softw.*, 29:39–63, 1995.

[3] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[4] USA Department of Defence. Military standard on software development and documentation (mil-std-498), 1994.

[5] Arie Van Deursen and Leon Moonen. Exploring legacy systems using types. In *Proc. of the 7th Working Conf. on Reverse Engineering*, page 32, Washington, DC, USA, 2000. IEEE Computer Society.

[6] Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Com. ACM*, 41(12):54–62, 1998.

[7] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November–December 2003.

[8] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. on Softw. Eng.*, 32(1):4–19, January 2006.

[9] M.E.C. Hull, K. Jackson, and A.J.J. Dick. *Requirements Engineering*. Springer, 2002.

[10] M. Lindvall and K. Sandahl. Practical implications of traceability. *Softw. Pract. Exper.*, 26(10):1161–1180, 1996.

[11] Marco Lormans and Arie van Deursen. Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 37–42, Long Beach, CA, USA, November 2005.

[12] Marco Lormans and Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proc. of the 10th European Conf. on Software Maintenance and Reengineering*, pages 47–56, Bari, Italy, March 2006. IEEE Computer Society.

[13] Marco Lormans, Hylke van Dijk, Arie van Deursen, Eric Nöcker, and Aart de Zeeuw. Managing evolving requirements in an outsoucring context: An industrial experience report. In *Proc. of the Int. Workshop on Principles of Software Evolution*, Kyoto, Japan, 2004. IWPSE04.

[14] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Enhancing an artefact management system with traceability recovery features. In *Proc. of the 20th IEEE Int. Conf. on Software Maintenance*, pages 306 – 315. IEEE Computer Society, 2004.

[15] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Can information retrieval techniques effectively support traceability link recovery? In *Proc. of the 10th Int. Workshop on Prog. Compr.*, Athens, Greece, 2006. IEEE Computer Society.

[16] Jonathan I. Maletic, Ethan V. Munson, Andrian Marcus, and Tien N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 47–54, Montreal, Canada, 2003.

[17] Hans W. Nissen, Manfred A. Jeusfeld, Matthias Jarke, Georg V. Zemanek, and Harald Huber. Managing multiple requirements perspectives with metamodels. *IEEE Softw.*, 13(2):37–48, 1996.

[18] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Trans. Softw. Eng.*, 20(10):760–773, 1994.

[19] Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Bjorn Regnell. A linguistic-engineering approach to large-scale requirements management. *IEEE Softw.*, 22(1):32–39, 2005.

[20] B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001.

[21] B. Ramesh, T. Powers, C. Stubbs, and M. Edwards. Implementing requirements traceability: a case study. In *Proc. of the 2nd IEEE Int. Symp. on Requirements Engineering*, page 89, Washington, DC, USA, 1995. IEEE Computer Society.

[22] Marco Toranzo and Jaelson Castro. A comprehensive traceability model to support the design of interactive systems. In *Proc. of the Workshop on Object-Oriented Technology*, pages 283–284, London, UK, 1999. Springer-Verlag.

[23] Antje von Knethen. A trace model for system requirements changes on embedded systems. In *Proc. of the 4th Int. Workshop on Principles of Software Evolution*, pages 17–26, New York, NY, USA, 2001. ACM Press.

[24] Antje von Knethen, Barbara Paech, Friedemann Kiedaisch, and Frank Houdek. Systematic requirements recycling through abstraction and traceability. In *Proc. of the Int. Conf. on Requirements Engineering*, pages 273–281, Washington, DC, USA, 2002. IEEE Computer Society.

[25] A. Zisman, G. Spanoudakis, E. Perez-Mi nana, and P.Krause. Tracing software requirements artifacts. In *Proc. of Int. Conf. on Software Engineering Research and Practice*, pages 448–455, Las Vegas, Nevada, USA, 2003.

TUD-SERG-2006-014
ISSN 1872-5392