Delft University of Technology
Software Engineering Research Group
Technical Report Series

An Industrial Case Study in Reconstructing Requirements Views

Marco Lormans, Arie van Deursen, and Hans-Gerhard Gross

Report TUD-SERG-2008-032

TUD-SERG-2008-032

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 *Empirical Software Engineering, 2009*.

An Industrial Case Study in Reconstructing Requirements Views ⋆

Marco Lormans[1], Arie van Deursen[1], and Hans-Gerhard Gross[1]

Delft University of Technology, The Netherlands
(M.Lormans, Arie.vanDeursen, H.G.Gross)@tudelft.nl

Abstract. Requirements views, such as coverage and status views, are an important asset for monitoring and managing software development projects. We have developed a method that automates the process of reconstructing these views, and we have built a tool, REQANALYST, that supports this method. This paper presents an investigation as to which extent requirements views can be automatically generated in order to monitor requirements in industrial practice. The paper focuses on monitoring the requirements in test categories and test cases. In order to retrieve the necessary data, an information retrieval technique, called Latent Semantic Indexing (LSI), was used. The method was applied in an industrial study. A number of requirements views were defined and experiments were carried out with different reconstruction settings for generating these views. Finally, we explored how these views can help the developers during the software development process.

1 Introduction

A "requirements view" on a system or development process offers a perspective on that system in which requirements assume the leading role [45]. A requirement view can be a combination of artifacts such as requirements and design information, showing how a requirement is transformed into a design artifact, and indicating how and where a requirement is covered by specific design artifacts, or where it is located in the system architecture. Examples are coverage views, such as "which design artifacts address which requirement?", or status views, such as "which requirements are already implemented?" The various requirements views help to avoid inconsistencies within the documentation of one kind of work product (requirements specification) or between the documentation of different types of work products (requirements specification and architectural design document) [13]. Requirements views help in improving the coherence between the work product documents, and lead to higher overall quality of the work products.

Requirements views are essential for successful project management, and for monitoring the progress of 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 of implementation issues, and more likely to be interested primarily in his requirements.

⋆ This is a substantially revised and expanded version of our paper [35]

Unfortunately, capturing, monitoring, and resolving multiple views on requirements is difficult, time-consuming as well as error-prone when done by hand [44]. The creation of requirements views necessitates an accurate traceability matrix, which, in practice, turns out to be very hard to obtain and maintain [12, 17, 18, 30, 49]. The tools currently available on the market, such as Telelogic DOORS and IBM Rational RequisitePro, are often not sufficient: keeping the traceability consistent using these tools is hard and involves significant effort [1, 34].

To remedy this problem, a significant amount of research has been conducted in the area of reverse engineering of traceability links from available software development work products [22, 36, 46]. Our own line of research has focused on the use of information retrieval techniques, in particular *latent semantic indexing* (LSI) [9], for this purpose, and on the application of the reconstructed matrices for view reconstruction, specifically. We incorporated our ideas in a method, called MAREV, and implemented the method in a tool, called REQANALYST [31, 32, 33].

While significant progress in this area has been documented, a number of open research issues exist, which we seek to explore in this paper. An initial question to be addressed is not related to the case study performed. It is about which requirements views are most needed in practice. To answer this question, a questionnaire was sent out to a dozen practitioners, and from the answers three important groups of views were distilled, which are described in detail.

As unit of analysis, one development project of LogicaCMG, an international IT services supplier, was scrutinized for the case study. The primary question addressed through this exploratory case study was how and to which extent requirements views can be reverse-engineered from existing work products. An important question hereby is whether the approach we proposed [31, 32, 33] may be used to reconstruct these views. To answer this question, it is described how our own prototype tool (REQANALYST) has been extended to support these views, offering project stakeholders means to inspect the system and development progress in terms of these views. Another question to be addressed through the case study is whether these reconstructed views can help in a real life software development process.

In the software development project under investigation in this case study, a traffic monitoring system (TMS) is developed, and it is outsourced to LogicaCMG. In the project, progress reporting to the customer must be done in terms of requirements, making accurate requirements views an essential success factor. The paper discusses the way of working in this project, and looks at how and to which extent reconstructed links can be used to support and enhance the way of working. In the case study, the focus lies on requirements views that are related to testing artifacts.

The remainder of this paper is organized as follows. Section 2 discusses existing work in the area of requirements views and reverse engineering of traceability matrices. Section 3 summarizes the methodology for generating requirements views, called MAREV. Sections 4, 5, and 6 present the requirements views aimed at, the way they are implemented in the REQANALYST tool, and the case study performed at LogicaCMG, respectively. The paper conclude withs a discussion, a summary of contributions, and suggestions for future research.

2 Related Work

2.1 System Views

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 [45] and mostly represent a specific perspective on a system. This perspective is often a subset of the whole system in such a way that its complexity is reduced. Each stakeholder is interested in a different part of the system. A stakeholder may be a developer who is only interested in a small part (a component, for example) of the complete system. The perspective that a view represents, can also be an abstraction of the system. It can give an overview of the whole system without providing too many details. Such a view from the top can be useful for a project manager or a system architect.

Nuseibeh *et al.* discuss the relationships between multiple views of a requirements specification [13, 45]. 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. It is based on the viewpoints framework presented by Finkelstein *et al.* [15]. This framework helps in organizing and facilitating the viewpoints of different stakeholders.

Zachman proposes "The Architecture Framework" focusing on information system views [60]. Hay uses the six views of this framework for requirements analysis [21]. In his approach, he uses the framework to define the requirements analysis process, which can be seen as the process of translating business owners' views into an architect's view.

The concept of a "view" also appears in other areas of software engineering such as architectural design. Kruchten introduced his "4 + 1 view model for architecture", where he defined five different concurrent perspectives on a software architecture [28]. Each view of this model addresses a specific set of concerns of interest to different stakeholders. Other examples are the "Siemens' 4 views" by Hofmeister *et al.* [23], the IEEE standard 1471 [25], and the views discusses by Clements *et al.* in their book "Documenting Software Architectures" [6, 7]. Van Deursen *et al.* also discuss a number of specific views for architecture reconstruction [11].

Finally, Von Knethen discusses view partitioning [57]. She considers views on the system, distinguishing, for instance, the static structure from the dynamic interactions in the system. These views support the process of impact analysis in two ways: they improve (1) the planning (estimating costs) as well as (2) the implementation of changes. Furthermore, the views allow the system to incorporate changes in a consistent way.

Although, much research has been done in the area of system views, there is no general agreement on what such views should look like, or which information they should contain. Every project setting seems to have its own specific information needs. Thus, views must be flexible in meeting these needs.

2.2 Document Standards, Templates and Reference Models

Another approach for separating concerns is to use a well structured document set, conforming to known templates such as MIL-std 498 [10], Volere [52], IEEE-std-830 [27],

or IEEE-std-1233 [26]. 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 they are difficult to keep consistent during development.

Nissen *et al.* show that meta-models help managing different requirements perspectives [44]. The meta-models define which information is available and how it is structured in the life-cycle. This comprises the development artifacts, including their attributes, and additionally, the traceability relations permitted to be set between these artifacts. If the information is not stored sometime in the life-cycle, it can never be extracted and used in a view. An important area of research is developing these meta-models [41, 50, 55, 57, 61], constraining the views to be generated.

Von Knethen proposes traceability models for managing changes on embedded systems [57, 58]. These models help estimating the impact of a change on the system, or help to determine the 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 [34]. 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 change constantly in big development projects [12].

2.3 Traceability Support and Recovery

Traceability support is required in order to reconstruct requirements views from project documentation. Several traceability recovery methods and supporting tools already exist, each covering different traceability issues during the development life-cycle. Some discuss the relations between source code and documentation, others address the relations between requirements on different levels of abstraction.

De Lucia *et al.* present an artifact management system, which has been extended with traceability recovery features [36, 40]. This system manages different artifacts produced during development such as requirements, designs, test cases, and source code modules. The Information Retrieval (IR) technique that De Lucia *et al.* use for recovering the traceability links is Latent Semantic Indexing (LSI). Furthermore, they propose an incremental traceability recovery process in which they try to identify the optimal threshold for link recovery in an incremental and iterative way [37]. The threshold determines which links should be considered as candidate links by a tool and which not.

Natt och Dag *et al.* [46] and Huffman Hayes *et al.* [22] use traceability reconstruction primarily for managing requirements of different levels of abstraction, such as reconstructing links between business and system requirements. Both, Natt och Dag *et al.* and Huffman Hayes *et al.*, have developed a tool to support their approaches. In [46], Natt och Dag *et al.* discuss their approach and tool, ReqSimile, that implements the basic vector space model which also forms the basis for latent semantic indexing. They report their experiences in [46], and the results are comparable to what we found.

In their tool called RETRO, Huffman Hayes *et al.* have implemented various methods for recovering traceability links [22]. 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 [5]. They are implementing their approach in a tool called Poirot [29]. They have also defined a strategy for discovering the optimal thresholds for determining candidate links [62].

Antoniol *et al.* [2] use information retrieval methods to recover the traceability relations between C++ code and documentation pages, and between Java code and requirements. Marcus and Maletic [42], and Di Penta *et al.* [48] use information retrieval techniques for recovering the traceability relations between source code and documentation. In addition, Di Penta *et al* [48] augmented their traceability approach with models of programmer behavior. The IR methods in these cases are mostly applied for reverse engineering traceability links between source code and documentation in legacy systems.

Marcus *et al.* [43] discuss how to visualize traceability links, and they introduce a tool, TraceViz, that implements their proposed requirements for traceability visualization. IR techniques are also used for improving the quality of the requirements set. Finally, Park *et al.* use the calculated similarity measures for improving the quality of the requirements specifications [47].

None of the discussed traceability reconstruction methods support the generation of requirements views for monitoring the requirements in the other work products. One reason for this is that current methods do not explicitly discuss the links that can be reconstructed and cannot be reconstructed. This makes it hard to define specific views and retrieve the information needed to manage a project with respect to evolving requirements.

3 MAREV and REQANALYST

In our earlier work, we have proposed an approach for reconstructing requirements views [31] and experimented with the reconstruction of traceability links in several case studies [32, 33]. The method is called MAREV: Methodology for Automating Requirements Evolution using Views. Besides that, the method has been implemented in a tool called REQANALYST. This section provides a brief overview of the tool as well as the underlying method.

3.1 MAREV: A Methodology for Automating Requirements Evolution using Views

MAREV consists of the following six steps (see also [31, 32, 33]).

Step 1: Defining the Traceability Meta-model. The underlying traceability meta-model defines the types of work products, such as business requirements, system requirements, design artifacts, or test cases, and the like, and the type of links that are

permitted within the development life-cycle. The choices made for defining the meta-model largely depend on the needs of the application domain. Examples can be found in [41, 50, 55, 57, 58, 61].

Step 2: Identifying the Work Products. The work products are identified in the provided project documentation or configuration management system, and mapped onto the traceability meta-model. Each work product is given a type and unique identifier if it has not already been assigned one. This unique identifier is a code plus a unique number, for example, a functional requirement description can have an identifier of the type "FR*xx*", where *xx* represents the number. This results in a set of functional requirement descriptions with the unique identifiers "FR01", "FR02", and so on. This step must be executed for every work product defined in the traceability meta-model. If requirements management tools such as Telelogic's DOORS are used, unique identifiers are provided automatically.

Step 3: Preprocessing the Work Products. The work products are preprocessed to support automated analysis for them. The text of each work product needs to be extracted and transformed into plain text. This step includes typical information retrieval activities such as lexical analysis, stemming, and so on.

Step 4: Reconstructing the possible Traceability Links The likely traceability links are reconstructed for which Latent Semantic Indexing [9] is used. The result of this step is the complete set of candidate traceability links.

Latent Semantic Indexing (LSI) is an information retrieval technique based on the vector space model. It assumes that there is an underlying or latent structure in word usage for every document set [9]. LSI uses statistical techniques to estimate this latent structure. A description of terms and documents based on the underlying latent semantic structure is used for representing and retrieving information. LSI starts with a matrix of terms by documents. Subsequently, it uses Singular Value Decomposition (SVD) to derive a particular latent semantic structure model from the term-by-document matrix. The result is a reduced model, the rank-*k* model with the best possible least square fit to the original matrix of terms by documents [9]. Subsequently, this model can be used to determine a similarity matrix.

Once all documents have been represented in the LSI subspace, the similarities between the documents can be computed. The cosine between their corresponding vector representations can be used for calculating this similarity metric. The metric has a value between $[0, 1]$ with a value of 1 indicating that two documents are (almost) identical. These measures can be used to cluster similar documents, or for identifying traceability links between the documents.

Finally, LSI does not rely on a predefined vocabulary or grammar for the documentation (or source code). This allows the method to be applied without large amounts of preprocessing (i.e., stemming) or manipulation of the input, and, therefore, it can reduce the costs of traceability link recovery considerably [36, 41].

Step 5: Selecting the Relevant Links. The possibly relevant links are selected automatically from the complete set of candidate links (from the LSI) using various link selection strategies. In our previous work, we proposed two link selection strategies, a *one* and a *two dimensional vector filter strategy* on the similarity matrix [33]. These link selection strategies combine the already known strategies *constant threshold* (represented by the symbol *c* in this paper) and *variable threshold* (represented by a percentage *q*) discussed by De Lucia *et al.* [36]. The one-dimensional filter strategy considers every single column of the similarity matrix separately. Each column vector of the similarity matrix is taken as a new set of similarity measures, and it combines, for each column, the constant and the variable threshold approaches. The two-dimensional filter strategy extends the one-dimensional strategy by considering both dimensions of the similarity matrix. The benefits of these strategies are that they guarantee certain level of quality by using the constant threshold, and, yet, they take only the best *k*% of the links for a certain work product. Both strategies are described in detail in [33], and they have shown improved results in terms of recall and precision. As with all information retrieval techniques, it is not guaranteed that all correct links are indeed found: both, false negatives, and false positives may arise.

Step 6: Generating Requirements Views. Finally, the requirements views are generated using the reconstructed traceability links. This step will be the focus for the rest of this paper.

Step 7: Tackling Changes. Finally, the reconstructed traceability links and generated requirements views need to be able to tackle changes in the requirements. Therefore, the validated traceability matrix and the newly reconstructed traceability matrix need be compared after each run of the MAREV approach. Users can then validate the impact of a requirements change in the traceability matrix.

3.2 The REQANALYST Tool Suite

In order to support the MAREV approach, we developed the REQANALYST[1] [33] tool. This tool can reconstruct traceability information and generate requirements views using that reconstructed traceability information. In this section we summarize our earlier work on REQANALYST. In Section 5 we focus again on generating our requirements views using REQANALYST.

Extract-Query-View Approach. REQANALYST adopts the Extract-Query-View approach used in many reverse engineering tools [56]. In this approach, first, the relevant data from the provided documents is extracted. This data, the work products, and, if available, the reference traceability matrices, are stored in a database. For reconstructing the traceability links, queries can be conducted on the database. The reconstructed information, combined with the data from the database, is used to generate the requirements views.

[1] REQANALYST is available from http://swerl.tudelft.nl/bin/view/Main/ReqAnalyst.

The reference traceability matrix is optional, and contains the correct links according to the experts in the project. It is only required to assess the outcomes of the tool, addressing the question as to which extent requirements views can be reconstructed automatically. Typical (reengineering) projects do not have such a matrix, to start with, and the ultimate goal is to generate this matrix automatically from the existing project documents, i.e., through using LSI.

Implementation. REQANALYST is implemented using standard web-technology. For storing the data, a MySQL database is used. It is implemented as a a Java web application using Java Servlets and Java Server Pages (JSP). For the case study, the Apache Tomcat 5.5 web server was taken for deployment.

Functionality. A REQANALYST session starts by logging in. Each user of REQANALYST has specific rights to see certain projects. After authentication the user gets a list of projects. Once the user has chosen a project, REQANALYST shows the main menu. This main menu follows the steps from the Extract-Query-View approach [56], including functionality for extracting the data from the complete set of provided documentation, and options for setting the parameters of the LSI reconstruction and the choice for a link selection strategy. Figure 1 shows a excerpt of the tool.

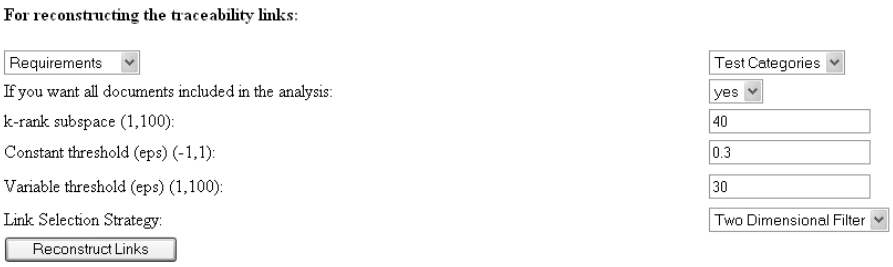

Fig. 1. Input screen for Traceability Reconstruction

Once REQANALYST has executed a reconstruction, a menu appears showing the reconstructed traceability matrix and a number of options for generating various requirements views. This menu shows all the metrics relevant for assessing the reconstruction, such as recall, precision, and the number of false positives and missing links in the traceability matrix. This menu is also used to generate the various requirements views.

Browsing in REQANALYST. An important feature of REQANALYST is the possibility to browse the reconstructed results. It allows engineers to inspect the reconstructed traceability matrix and browse through the traceability links, implemented as hyper

Questions:	
1)	What is your role in the software development life-cycle?
2a) 2b)	What do you expect from a requirements view? What information would you like to see in a requirements view? (Examples: coverage, functionality, status)
3a) 3b)	What do you think persons in the roles below expect from a requirements view? - Project Manager - Requirements Engineer - System Architect - Programmer - Test Engineer - Quality Manager - Other? (please also define the role) What information do you think they would like to see? (Do not fill in your own role again)
4)	Do you think it is feasible to extract this information from the work products currently produced during development? (requirements specifications, design documents, etc.)

Table 1. Topics put forward in the Questionnaire

links. When following the hyper link, all the information concerning the two entities involved becomes available and can be inspected. For example, the original text of both entities is shown in one view.

Furthermore, the reconstructed matrix can be compared with a reference matrix, if available. The reference matrix represents the traceability matrix as determined by the developers of a system and is only required for evaluation purposes. The correctly reconstructed links (correct positives) are indicated with an "X" and the cell is colored green. The false positives are indicated as "fp" and are colored yellow. Furthermore, the false negatives (missing links) are indicated through "fn" and are colored red.

4 Which Views are Needed in Practice?

While MAREV and REQANALYST provide a method and tool support for obtaining requirements views, it is less obvious which requirements views are actually needed in practice. To address this issue, we have set up a questionnaire and distributed it among various practitioners. Below, the questionnaire is described, and the three main types of views that emerged from our survey are discussed.

4.1 Requirements View Questionnaire

The goal of our questionnaire was to get an impression about which views would be helpful and which information these views should represent. The questions asked to the participants are shown in Table 1. The questionnaire was distributed among people of various roles within the software development life-cycle. The roles 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 project in which we are involved[2]. 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 involved, all of which provided many replies according to their various roles. In total we had 12 fully filled in questionnaires containing around 100 descriptions of desirable views for different roles in the life-cycle.

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

4.2 Main Outcomes

The outcome from the questionnaire is that requirements should be able to be traced into their associated subsequent work products. A challenge in that respect 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 address this issue, and make it easier to deduce the right information needed for the view in question.

Another lesson learned from the questionnaire is that the following information is desirable in a requirements view:

- For each requirement, the source, description, motivation, importance, history, status and dependencies to other work products. This is actually an obligation of the new safety standard ISO/WD 26262 for systems in the automotive domain that is currently being developed [14].
- 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. 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 it was concluded 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 three types of views were distilled: *Coverage* views, *Life-cycle Path* views, and *Status* views. Below, they are discussed in detail.

[2] www.merlinproject.org

4.3 Coverage Views

Requirements *coverage* views focus on the localization of the requirements in the rest of the system. These views show whether and where a certain requirement is associated with another artifact in the system. This can be coverage in the system architecture, in the detailed design, or in the test cases, to name only a few instances. The number of different types of coverage views depends on the meta-model defined for the development process. It prescribes which phases are defined and which work products are produced during these phases. This view is often used for tracing requirements changes into subsequent work products [54, 57], and it can, therefore, be used for impact analysis in system evolution [4].

According to Costello *et al.*, requirements coverage is defined as: *The number of requirements that trace consistently to the next level up or down* [8]. They 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 [24]. 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, for example a test case, exists, and this link is correct, then is the requirement covered by that work product. The requirements coverage view shows 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, the percentage of requirements (compared to all requirements) covered by a test case can be defined 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 fundamental, and can be used for requirements coverage in other life-cycle phases as well, such as the coverage of requirements in the design.

4.4 Life-cycle Path Views

Requirements *life cycle path* views deal with the transformations and decompositions that a requirement undergoes throughout the development process. The questionnaire showed 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, it becomes apparent that these are the horizontal and vertical dimensions of this life-cycle model.

The length of a life-cycle path is captured by the second traceability metric of Hull *et al.*, called *Traceability Depth* [24]. This metric relates to the number of layers along which the traceability extends, for example the layers along the left leg of the V-model

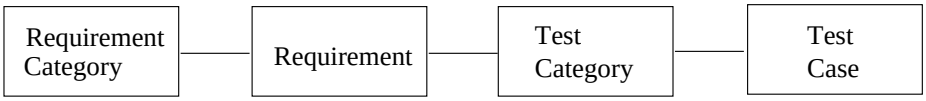

Fig. 2. An example of a life-cycle path

for capturing software development. It can also be seen as the number of (model) transformations between the different types of work products.

As an example, Figure 2 shows a Requirements–Test-Path in a traceability meta-model. This example is taken from our case study which will be discussed in Section 6. It shows that the focus of interest lies in following the path of the requirements categories, via requirements and test categories, to test cases. The path extends along 4 layers according to Hull *et al.* Note, that a coverage view addresses only one layer.

In order to further characterize a life-cycle path view, another metric from Hull *et al* is relevant as well. This other metric, called *Traceability Growth*, measures how a requirement expands down through the layers of the meta-model (in our case the life-cycle path) [24]. For example, a requirement can be covered by one test case or by multiple test cases. This is also a useful metric for impact analysis, which is why we will include it in our life-cycle path view.

4.5 Status Views

Requirements *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. In other words, if a link exists from a requirement to a source code document, it can be assumed that the status of the requirement is “implemented”. In addition, this information can be used in order to obtain a coverage measure for the number of implemented requirements for project management purposes. For example, status views may be associated with a measure expressing that 60% of all requirements have the status “implemented”. A project manager can use this information to monitor the progress of the project. Other management information can be obtained by computing percentages of requirements that have reached a certain status such as “tested successfully”.

Traceability support is often not enough to generate complete status reports of requirements, for example, when a project manager needs to know whether 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 whether the implementation of a specific requirement has passed its test or not. This information needs to be extracted from the document and included in the status view as well.

In the case study, this extra status information was monitored in addition to the normal traceability data. We tried 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".

5 Implementing the Views in REQANALYST

The three views presented 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. This section discusses how our REQANALYST tool as described in Section 3.2 has been extended to incorporate support for these three views.

Coverage Views. The "Coverage View" as implemented in REQANALYST shows the number of requirements that are covered (linked correctly) by some other work product, and the total number of requirements that are analyzed. It also shows the coverage percentage as defined in Section 4.3, i.e., percentage of the correctly reconstructed links between requirement and associated other work product. Furthermore, 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. We call this view the "Orphans View". This view shows the same results as the coverage view, except for the related artifacts: as there are none, these cannot be shown. This view is important for developers as they need to inspect why the requirements in this view are not yet covered in the system.

Life-cycle path Views. The "Life Cycle Path View" as implemented in REQANALYST displays the stages involving a requirement. In particular, a tabular view is shown, illustrating the work products a requirement is related to, such as requirements categories or test cases. This table can also be used to obtain the values for the traceability growth metric at the various levels in the life cycle path. An example for our case study based on the traceability model in Figure 2 is shown at the end of the paper in Figure 5.

Status Views. The "Status View" as implemented in our REQANALYST tool is based on the observation that every entity of a work product type can have multiple status attributes attached to it. So, besides extracting the relevant data for executing the automated reconstruction, it can also extract the additional status attributes from the provided documentation. These status attributes are saved separately in the database. When a user generates a view of a specific "requirement – test case" relation, for instance, it can also show the status attributes concerning this relation.

6 Case Study: LogicaCMG

The previous sections discussed the three most essential views considered by engineers, and we have proposed a method and a tool for reconstructing these views automatically

from the available work products. This section presents the case study performed at LogicaCMG aimed at illustrating how the method and the tool work out in practice.

We begin with laying out the case study design, making use of the guidelines provided by Yin [59]. Then, after discussing the nature of the project and the development process followed, we describe which requirements documents we used as input for the reconstruction effort. Furthermore, we explain the reconstruction approach and its specific parameter settings used, followed by a discussion of the traceability matrices obtained. Finally, we discuss how these matrices lead to the requirements views considered.

6.1 Case Study Design

The study aims at answering the following two essential research questions: (1) How and to which extent can requirements views be reconstructed from existing work products, and if this is the case, (2) can these requirements views help during development? Addressing question (1), we believe requirements views can be reconstructed, although, not up to the level desired. So, the question remains, whether the proposed techniques, although sub-optimal, may have a positive effect on the overall development process of a software project. The unit of analysis is a large and long-lasting development project carried out by LogicaCMG which is described in much more detail below. Question one is assessed by typical measures used in traceability link reconstruction, i.e. recall and precision. Additional measures are used to indicate the likely effort to assess the reconstructed views, i.e. validation percentage, and coverage. Addressing question (2) is a lot more difficult, because comparable data for a fully manual reconstruction approach are lacking. In that respect, we cannot come to definite objective conclusions on the performance of the automatic approach for the task under consideration.

6.2 Case Study 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 that is required to operate at its maximum capacity. 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. The TMS owners decided to outsource the development of TMS to LogicaCMG.

Initially, LogicaCMG used IBM Rational RequisitePro for managing the requirements and MIL-std-498 [10] for documenting their work products. The project consumed 21 man years in the past 3 years of development. In total, there are over 1200 requirements and over 700 test cases. All the traceability links between the work products were manually set. This manual effort, which is time-consuming and error-prone, is acceptable if it is done once. However, when existing requirements evolve 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. These are examples for why tracing becomes inconsistent, and must be redone, eventually. Sometimes, the large number of

changes made that the effort needed for updating the traceability links was comparable with completely resetting all the links. Having an automatic technique in place to reconstruct the inconsistent traceability links may, thus, save a lot of effort.

Furthermore, the customer was not willing, initially, to operate on the tagged documentation LogicaCMG provided along with the tool, since the customer wanted to keep control of 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. Some of the mechanisms used for managing requirements evolution in this setting are described in our earlier work dealing with the same case study [34].

This way of working had two important shortcomings. First, it made the information used for monitoring the progress of the requirements during the development process unreliable. This was mainly due to the difficulty of keeping the traceability links consistent during the evolution of the project. This increased the risks during the integration phase, such as requirements that are not implemented, or functionality that should not be implemented in the system. Second, the manual work for synchronizing the updates from the client introduced errors, and was time-consuming.

In a later stage of the project, the customer dropped the demand of ownership of all documents. Furthermore, LogicaCMG decided to reduce the number of links maintained to the most essential ones. In particular, test documentation and test descriptions were merged, thus simplifying the underlying meta-model. This reduction of possible traceability links also helped to reduce the risk of inconsistencies.

In addition to that, part of the traceability matrix was maintained within the documentation itself, instead of in a separate spreadsheet. Test documents include the unique identifiers of the requirements they cover. The documents are structured in such a way that the Doxygen[3] documentation generator can be used to produce an HTML representation of the full matrix.

In both, the initial, and the current way of working, traceability links are set manually. Our approach aims at offering partially automated tool support for this. The case study at hand offers an opportunity to investigate whether our proposed approach can be useful in practice, and whether it may reduce the effort needed for consistent traceability support. In the case study, only the current way of working will be considered.

6.3 Available Data

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 they are structured according to MIL-std-498 [10]. This means that traceability data is incorporated in these documents and that it is possible to obtain a reference traceability matrix from this data.

[3] www.doxygen.org

Work product type	Number	Size in terms	Avg. terms per doc.
Requirements Categories	45	1168	183
Requirements	121	695	29
Test Categories	29	589	183
Test Cases	98	886	107

Table 2. TMS Case Study Statistics

Besides the two MS-Word documents, an HTML document generated by Doxygen is available. This document is an addition to the STD, and it contains the description of the test cases. It also comprises the description of the test categories and, in some cases, also the descriptions of the requirements it refers to (see Section 6.2). Doxygen uses this additional information of the test categories, and, if available, the requirements to generate the HTML document. The HTML document is accompanied by an MS-Excel spreadsheet, which contains the traceability links between the requirements and the test cases. For our LSI analysis, we only extracted the test case descriptions without the additional data (as this data is sometimes missing).

Our meta-model for this case study is shown in Figure 3. It consists of the following work products. In the SSS, a hierarchy of requirements 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, so they were taken into account for analysis as well.

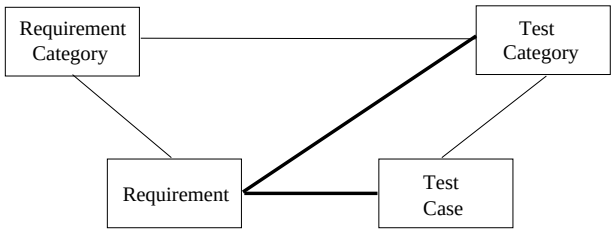

Fig. 3. Traceability Meta-Model. The bold lines indicate the explicit traceability links available in the study.

Examples of requirements categories are general ones, such as goal and domain, as well as 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. The traceability between the requirements categories and requirements can be derived from the hierarchy. This traceability is not incorporated explicitly in the MS-Word documents.

For the test cases, the same hierarchy can be identified, resulting in the separate work products “test category” and “test case”. Both are uniquely identifiable in the pro-

vided documentation. The main difference is that the two documents are not related directly, but only through the requirements. Thus, the individual test cases are not identifiable in the STD. In order to work out the hierarchical relations, the HTML files that include the test case descriptions and test scripts, have to be checked. They contain an identifier of a test category in the STD. However, the STD does contain the traceability links between the requirements and the test categories.

The progress of 121 requirements, distributed over 45 categories, was monitored. As these requirements are provided by MS-Word documents, some manual processing had to be done, in order to extract the relevant data from the SSS and store the processed tokens of text in the database. The requirements consist of a unique identifier and a description. Besides the requirements, the SSS document contains some context explaining certain domain knowledge for a group of requirements. This data was extracted as well and stored in the database, marking it as "context".

For the other work products, the requirements categories, test categories and test cases, the same approach for obtaining the relevant data was used, resulting in 45 requirements categories, 29 test categories and 98 test cases (see Table 2, above).

LogicaCMG presently maintains two types of links, as indicated by the bold lines in Figure 3. These links between requirements and test cases, and between requirements and test categories, are maintained in the SSS, STD, and spreadsheet documents. The remaining links in Figure 3 can either be derived from the maintained links, or from the hierarchical structure of the documents.

Link source	Link target	# Reference links	# Candidate links
Requirements Categories	Requirements	121	5445
Requirements Categories	Test Categories	31	1305
Requirements	Test Categories	110	3509
Requirements	Test Cases	297	11858
Test Categories	Test Cases	122	2842

Table 3. Number of reference links and candidate links in the TMS case study

Table 3 displays, for each link type, the total number of candidate links that can be reconstructed as well as the total number of links in the reference traceability matrix. For example, there are 297 reference links derived for the "requirements – test case" link, whereas the total number of candidate links is $121 \times 98 = 11858$. The objective of our approach is to find this small number of correct reference links in the complete set of candidate links.

6.4 Reconstruction Approach

Reconstruction Input Parameters. The reconstruction of the traceability matrices for the different link types can be tuned in several ways. As we will see, the various link types call for slightly different parameter settings.

In all cases, we adopt a rank-k subspace of 40%. This is the size of the reduced semantic structure model produced by the singular value decomposition step of LSI. The new matrix is only 40% of the size of the original matrix, and, in LSI, it is important for filtering out unimportant details, while keeping the essential latent semantic structure intact. This step of LSI can be regarded as compressing the same information in a smaller sub-space [19], thereby generalizing the information contained.

The constant threshold is set to $c = 0.3$, i.e., two documents with a similarity below this value of c are never related. The variable threshold q is varied between 20% and 80%, indicating that the best q% of the interval between the minimum and the maximum of the similarity measures for a given document are used. The question here is which links are indeed relevant, or, in other words, where do we draw the line between interesting links and irrelevant links [20]?

These parameters are chosen according to our experience in applying LSI (see [33] for details on these parameters), and in the future, we anticipate that further "rules of thumb" for adjusting these parameters according to the problems at hand will have to be devised. In the presentation of the results in Tables 4–8 the first two columns indicate the values of c and q used.

Obtaining the Reference Matrix. The traceability data maintained manually by the software engineers at LogicaCMG were used as reference matrices in our case study. Maintaining such matrices and keeping them consistent by hand is hard and error-prone (see Section 6.2), so that the matrices were validated once more by Logica engineers. The existing matrix was compared with a matrix obtained automatically using our LSI-based approach. Assessing 100% of the links was considered too time-consuming. A rank-k subspace of 40%, $c = 0.3$, and $q = 20\%$ was used as inputs for this comparison. The engineers worked about 30 minutes to inspect the 29 false positives and 59 missing links issued by the tool (see Table 4). This resulted in resetting four missing links. Initially, they were indicated as link in the original matrix, but because REQANALYST did not reconstruct them, the engineers reassessed the links and decided to remove them from the reference traceability data. This improved the traceability matrix used as reference in our other reconstruction results.

Reconstruction Output Parameters. For each of the reconstructed matrices in Tables 4–8, seven results are shown that help to assess the usefulness of the reconstruction approach.

The set of *reconstructed links*, generated by REQANALYST, consists of *correct positives*, which are correctly reconstructed compared to the reference traceability matrix, and *false positives*, which are incorrectly reconstructed compared to the reference traceability matrix. Next, the *missing links* are shown (also known as false negatives), which are the links not reconstructed by REQANALYST, but identified as links according to the reference traceability matrix.

Finally, two commonly used metrics in the area of information retrieval are depicted; *recall* (correct positives / total reference links) and *precision* (correct positives / total reconstructed links) [3, 16, 51, 53]. The ultimate goal would be to achieve a recall of 100% and a corresponding precision that is as high as possible, since in that case we

only need to *eliminate* false positives. A recall below 100% which is often the case [42], inevitably means there are also false negatives (missing links). In the worst case, all candidate links need to be checked to identify these missing links, which takes much effort, but one of the goals of our approach was to reduce the manual effort needed to support consistent traceability (see Section 6.2).

Besides these metrics two other metrics were calculated, the *percentage of validation work* and the *coverage percentage*. For the application shown, the results of these last two columns 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). A validation percentage of 2% (see first row Table 4) means that the developers only need to validate 2% of all the candidate links manually.

The *coverage percentage* establishes a connection between the traceability matrix and the coverage views discussed in Section 4.3. The *coverage percentage* refers to the percentage of correctly covered work products compared to the total number of work products of that particular type, for example, the total number of correctly covered requirements compared to all the requirements.

6.5 Reconstructed Traceability Matrix Results

Given the traceability meta-model from Figure 3, five traceability link types are possible. First, we discuss the quality of reconstruction results for the link types LogicaCMG maintained, "requirements – test categories" and "requirements – test cases". Next, we discuss the link types we derived indirectly, "requirements categories – requirements", "test categories – test cases", and "requirements categories – test categories".

"Requirements – Test Categories". Table 4 shows the results for the link reconstruction between the requirements and test categories. When we increase q we see the recall increasing and the precision decreasing as expected. The validation percentage also increases, meaning more links need to be validated. A low validation percentage is positive, as it indicates the effort needed to keep the traceability support consistent after a change, for example. In the case of $q = 20\%$, only 2% of the total candidate links need to be validated. In this example, 98% of the candidate links do not need to be validated.

However, in the case where the validation percentage is 2%, there are also 59 correct links missing compared to the reference traceability matrix. We would like to achieve a recall of 100%, so that only false positives need to be eliminated (see Section 6.4). Table 4 shows that with a constant threshold of $c = 0.3$, we never achieve a recall of 100%. Therefore, c was decreased to 0.2 and 0.1. With $c = 0.1$, a recall of almost 100% can be achieved. Unfortunately, the number of false positives increases, and, accordingly, the validation percentage. Yet, the total effort reduction is $100 - 64 = 36\%$. Antoniol *et al.* [2] used a similar effort estimation, which they called the Recovery Effort Index (REI). It is not clear, however, whether such measurements are realistic indicators of effort, because of lack of empirical data about a manual traceability recovery process. This will require more comparative studies in the future.

Link Type: Requirements – Test Categories								
c	*q*	Reconstructed Links		Missing Links	Recall	Precision	Validation Percentage	Coverage Percentage
		Correct Positives	False Positives					
0.3	20%	51	29	59	0.46	0.64	2	43
0.3	40%	75	324	35	0.68	0.19	11	62
0.3	60%	82	722	28	0.75	0.10	23	68
0.3	80%	82	740	28	0.75	0.10	23	68
0.2	80%	95	1389	15	0.86	0.06	42	77
0.1	80%	107	2152	3	0.97	0.05	64	83

Table 4. Reconstruction results for links between requirements and test categories

From these results it can be concluded that it is very hard to recover the last 10–15 missing links with the approach presented, and realize a recall of 100%. It is an open question whether there are textual revisions to the documents conceivable (such as an annotation mechanism, or more consistent wording of the requirements) that would enable automatic recovery.

The final 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 coverage percentage will get closer to the coverage that is obtained from the reference matrix. In the TMS case study, 85% of the requirements are covered by test categories. The missing links cause the coverage percentage to be 83% instead of 85%, as expected.

Link Type: Requirements – Test Cases								
c	*q*	Reconstructed Links		Missing Links	Recall	Precision	Validation Percentage	Coverage Percentage
		Correct Positives	False Positives					
0.3	20%	66	419	231	0.22	0.14	4	26
0.3	40%	141	2254	156	0.48	0.06	20	45
0.3	60%	186	3938	111	0.63	0.05	35	53
0.3	80%	186	3967	111	0.63	0.05	35	53
0.2	80%	223	6265	74	0.75	0.03	55	67
0.1	80%	260	8508	37	0.88	0.03	74	74
0.05	80%	265	8682	32	0.89	0.03	75	74
0.05	90%	276	10030	21	0.92	0.03	87	74

Table 5. Reconstruction results for links between requirements and test cases

"Requirements – Test Cases". Table 5 shows the results for the links between requirements and test cases. The results are of lower quality compared to the links between

requirements and test categories: For every value of the variable threshold q, the recall and precision are lower in this case. In order to get a reasonable recall, we need to decrease the constant threshold to $c = 0.05$. Even then, the recall is not 100%: again, we are not able to recover the final 20–30 missing links, which, are indicated as traceability links in the reference matrix

This result has consequences for the applicability of this link relation. Looking at the validation percentage, in can be observed that 87% of all candidate links need to be validated. This means that many false positives have to be eliminated and almost all (87%) of the links must be checked manually. Somehow, there seems to be a mismatch between the requirements and the test cases.

The coverage of requirements in test cases also confirms this mismatch. The coverage percentage is 79% in the reference traceability matrix. Our result approaches that value, as expected. But, comparable to the previous case, some requirements seem to be hard to link to test cases as indicated by the difference between our value of 74% and the reference value of 79%.

A way to improve the results can be by incorporating the additional information of the test categories and requirements in the LSI analysis. This was not done, since this information is missing for some of the test cases (see Section 6.3). By adding this information, the identifiers of the test categories and requirements can be included in the LSI analysis, causing the similarity value to increase. The test categories did contain the unique identifiers of the requirements in their descriptions. This is probably one of the reasons why the results for the links between the requirements and test cases is lower. It also demonstrates the importance to include the identifiers in the LSI analysis.

"Requirements Categories – Requirements". As discussed in Section 6.3, the System/Subsystem Specification (SSS) consists of a hierarchy of requirements. The higher level structure of requirements is called requirements categories. We investigated whether this containment relation can be identified using the link reconstruction approach presented.

Link Type: Requirements Categories – Requirements								
c	q	Reconstructed Links		Missing Links	Recall	Precision	Validation Percentage	Coverage Percentage
		Correct Positives	False Positives					
0.3	20%	91	32	30	0.75	0.74	2	75
0.3	40%	113	313	8	0.93	0.27	8	93
0.3	50%	118	699	3	0.98	0.14	15	98
0.3	60%	119	1300	2	0.98	0.08	26	98
0.3	80%	119	1754	2	0.98	0.06	34	98

Table 6. Reconstruction results for links between requirements categories and requirements

Table 6 shows the results for the links between the requirements categories and requirements. These results are promising. Except for the three missing links, we already

realize a recall of almost 100% with $q = 50\%$. None of the previous results has shown such high quality.

This result can be explained by the fact that a requirements category consists of one or more requirements *plus* some extra context. So, the requirements descriptions can literally be found in the description of the requirements category. The extra context is, in most cases, a general description of the requirements category. Our reconstruction approach benefits directly from the fact that a requirements category contains the individual requirement descriptions.

When doing a qualitative analysis on the three missing links we find a plausible explanation for the fact that they are not reconstructed. The two links we could not reconstruct are canceled and they have no text describing the requirements except for the statement "canceled".

As a consequence from the current configuration, the effort needed to validate the links is low. Besides that, the coverage is almost 100%. This means that all the requirements are covered by a requirements category. The reference matrix also shows that all the requirements are covered by a requirements category. Our reconstruction results confirm this (see the last column of Table 6).

"Test Categories – Test Cases". The same analysis that was done for the requirements categories and requirements was also carried out for the test categories and test cases. The major difference with the requirements hierarchy is that the test categories do not contain the test cases. The test categories are described in the Software Test Description (STD) and the test cases are described in the generated HTML document. There are no reference links maintained by LogicaCMG for this relation, so these links had to be derived via the links of the requirements.

Table 7 depicts the results of the link between the test categories and test cases. The results are comparable with the results of the reconstruction between the requirements and test cases. Again, it is difficult to realize a recall of 100%, so that the constant threshold must be decreased, which leads to almost 20 links not being recovered by the tool.

Link Type: Test Categories – Test Cases								
c	q	Reconstructed Links		Missing Links	Recall	Precision	Validation Percentage	Coverage Percentage
		Correct Positives	False Positives					
0.3	20%	43	73	79	0.35	0.37	4	62
0.3	40%	62	324	60	0.51	0.16	14	62
0.3	60%	71	755	51	0.58	0.09	29	66
0.3	80%	71	867	51	0.58	0.08	33	66
0.2	80%	101	1512	21	0.83	0.06	57	83
0.1	80%	105	1682	17	0.86	0.06	63	83
0.05	80%	105	1682	17	0.86	0.06	63	83

Table 7. Reconstruction results for links between test categories and test cases

With a recall value of 86% already 63% of all candidate links need to be validated. If the aim is to achieve a recall of 100%, probably all candidate links need to be validated. This makes the effort reduction for this reconstruction minimal. In the future, we will have to find ways to increase recall without sacrificing precision.

The coverage of the reference matrix is 83%. A recall of 86%, realises a coverage of 83% which is equal to the coverage value of the reference matrix. This can be explained by the definition of the coverage metric. The coverage metric takes into account individual requirements. It checks if a test category is covered in the other work product, that is, the test cases. Thus, it only needs one consistent link to a test case to be set as covered. Still, a test category can have multiple links to multiple test cases. If one of these "extra" links is not reconstructed, this does not influence the coverage metric. Again, this metric only needs one consistent traceability link.

"Requirements Categories – Test Categories". We expected the results of the link between the requirements categories and test categories to be comparable with, or even better than the relation between the requirements and the test categories. This has the following reasons. First, the level of granularity should match better. Earlier results show that if there is a mismatch in the level of granularity, the reconstruction results of LSI will decrease [32, 33]. Second, the requirements categories contain more text, so the vector representation of the requirements categories devised during latent semantic analysis is expected to contain more terms than the one for the requirements.

Table 8 shows the results of this link type. The results are indeed comparable with the results depicted in Table 4, but the results are not better. Thus, the clustering of the requirements into categories does not imply an improvement of the results. In other words, the "richer' vector representation of the requirements categories (because of the larger text size), does not influence the vector representation of the requirements in a positive way, compared to the vector representation of the test categories. The vector representations of the requirements and the requirements categories are comparable, causing the similarity measure to be comparable.

Link Type: Requirements Categories – Test Categories								
c	q	Reconstructed Links		Missing Links	Recall	Precision	Validation Percentage	Coverage Percentage
		Correct Positives	False Positives					
0.3	20%	15	17	16	0.48	0.47	2	52
0.3	40%	17	85	14	0.55	0.17	8	59
0.3	60%	20	212	11	0.65	0.09	18	69
0.3	80%	21	224	10	0.68	0.09	19	72
0.2	80%	27	564	4	0.87	0.05	45	90
0.1	80%	31	795	0	1.0	0.04	63	90

Table 8. Reconstruction results for links between requirements categories and test categories

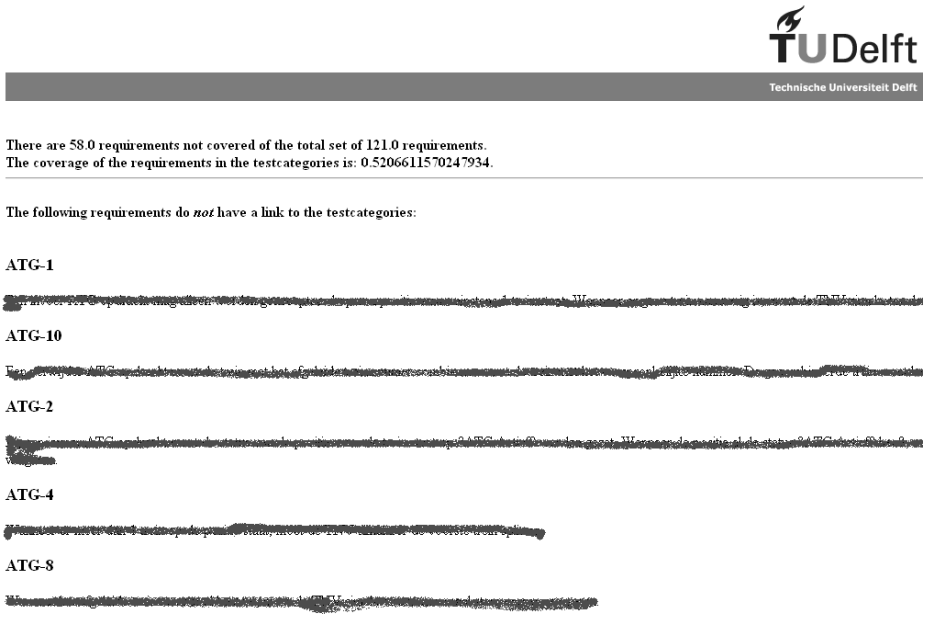

Fig. 4. Reconstructed coverage view. Company sensitive details have been made illegible on purpose.

6.6 From Traceability Matrices to Requirements Views

The previous section presented the reconstruction results of the different traceability link types. The generated views were used to fill in the last two columns of the Tables 4, 5, 6, 7, and 8. The other metrics such as recall and precision are not relevant for the users of REQANALYST, and, thus, will not be depicted in a requirements view. Each view can be tailored to the needs of the users.

Figure 4 depicts an example of a coverage view. This view shows the number of requirements that are not covered in the test categories. In this case, 58 requirements are not covered and this results in a coverage of 52%. This view also lists each requirement that is not covered. The user can scroll this list and take the appropriate action.

The views can use the automatically generated traceability links or the reference traceability matrices stored in the database. Finally, a validated matrix can be stored in the database as well. This validated matrix is then the preferred option for generating the views. To create a validated traceability matrix, all the reconstructed links are listed. The expert can review the complete list of reconstructed links and confirm or decline each candidate link. The links that are confirmed form the validated traceability matrix.

In order to create the Life-Cycle Path views, we can either use the reconstructed traceability data, or the reference traceability data. Figure 5 shows an example of a life-cycle path view, in which the requirements categories assume a leading role. We have made the identifier unreadable for confidentiality reasons. Figure 5 only shows a subset

of 4 requirements categories. As can be seen, each requirements category results in 3 or more requirements. The last requirements category even results in 30 requirements. Next, the several requirements are again captured in one or more test categories. Note that in this case, the *traceability growth* is less than 1 (more artifacts on the lower level, than on the higher level). The first 3 requirements are captured in 1 test category, and the 30 requirements are captured by only 5 test categories. Finally, the traceability growth between the test categories and the test cases is greater than 1 (more artifacts on the lower level than on the higher level). The 5 test categories are covered by 27 test cases, and the 1 test category is covered by 3 test cases. The first two test categories do not have test cases related to them.

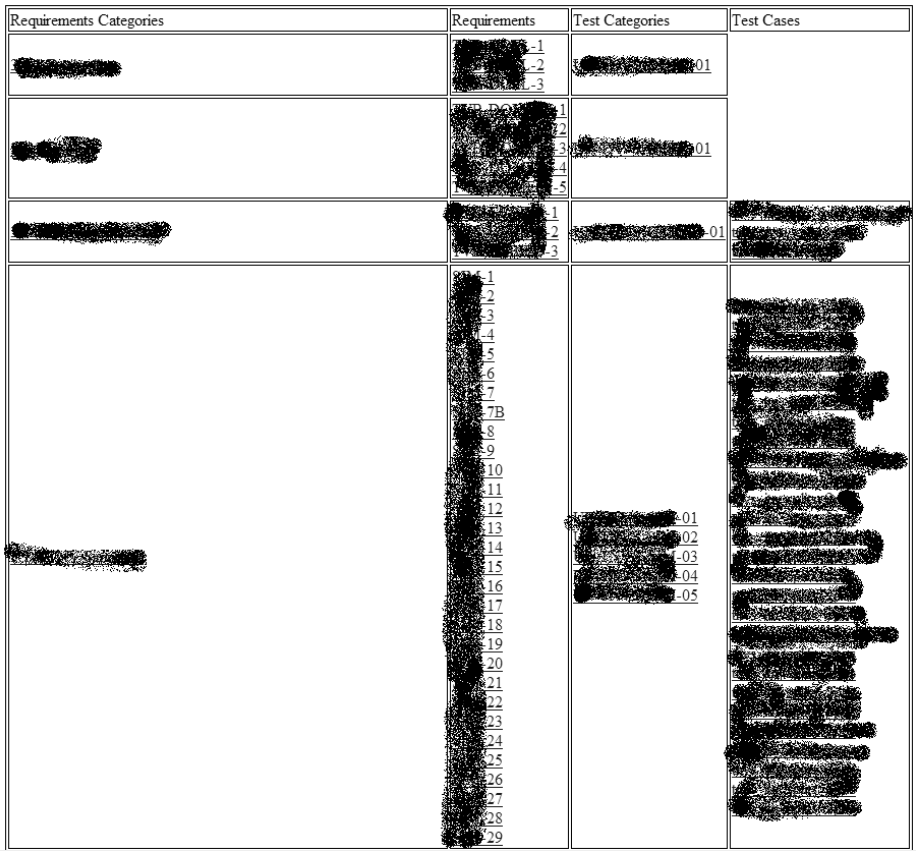

Fig. 5. Reconstructed life-cycle path view. Company sensitive details have been made illegible on purpose.

Finally, we are not able to show an example of a status view. In this case study, the status attributes are not provided in the documentation. So, we cannot show whether

a requirement is approved, or whether, a test case is executed and the system passed the test. Future case studies should provide this information. If status attributes are maintained, this additional information can easily be incorporated in a life-cycle path view.

7 Discussion

Quality of the Reconstructed Links The LogicaCMGcase study, demonstrates that the results for the various link types differ:

- Linking requirements to test *categories* worked out reasonably well. This is an important link type, maintained manually by LogicaCMG.
- Linking requirements to individual test *cases* was harder: apparently the test case descriptions are too short and too specific to link them easily to requirements prose.
- Linking requirements to their *requirements* category worked out very well, thanks to the fact that the requirements text was included in the category description.

Consistent Traceability Support. Our analysis identified several small inconsistencies. 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 canceled, was still included in the traceability data. The manual synchronization of these work products is, apparently, error-prone. REQANALYST can identify these inconsistencies, so that the developer can correct them. In this way, maintaining consistent traceability support becomes easier.

Requirements Views. Although more views can be defined in REQANALYST, the current views already got positive feedback from the developers at LogicaCMG. Our views increase developers' insights in the system and they improve the possibilities to review and validate the requirements systematically. Individual requirements can be inspected with respect to their coverage and their role within the system, using the life-cycle paths. Therefore, not all possible related documents need to be checked completely, reducing validation effort.

Currently, the number of views that can be generated using Doxygen is limited. The hyper links Doxygen is able to generate from its input files are bound to the information that is captured in those files. Our approach is more flexible. With the reconstructed traceability data we can generate the same and additional views compared to the Doxygen approach. So, our approach extends the current way of working at LogicaCMG.

An issue is the fact that our views greatly depend on REQANALYST's traceability support (as discussed above). Once the traceability is consistent, monitoring the progress of the requirements is improved by the requirements views proposed.

Effort Reduction. It is difficult to estimate whether and to which extent 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) [37, 39]. Following increments can take into account the validated reference traceability matrix. So, false positives that are already discarded from a previous reconstruction, and that do not relate to a change, are ignored. We expect that this will, again, reduce the effort for doing a next update. Only a small number of links, the links that are concerned with the changes, need to be validated. Initially, our reference traceability data was updated manually after the validation session, together with the expert. In order to come to final conclusions, in the future, we will have to pay more attention to how people are really constructing traceability links manually, and compare that to the performance of our automatic method.

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 in order to enable full automated traceability with a 100% recall. If projects could improve their documentation, for example along the lines proposed in [38], and that would enable fully automated traceability reconstruction, the benefits for practice would increase considerably.

Reconstruction Technology. The case study shows that in order to get a high recall, we have to live with a rather low precision – figures which are consistent with earlier studies [32, 37]. This raises the question whether the information retrieval approach used, latent semantic indexing, can be further refined. Future work is needed to determine whether there are specific characteristics of the requirements specification domain that can help to obtain better results. For example, the hierarchical nature of requirements documents may offer further clues for reconstructing links.

In addition to that, the specific link *selection* approach could be further refined. Presently, we made use of our two-dimensional link selection strategy as described in our earlier work [32, 33], since in a set of separate case studies this strategy performed best. It may be worthwhile to investigate alternatives to this approach, possibly differentiating between various link types.

Generalizing the Findings. Naturally, many of the details in the case study are specific for the setting at LogicaCMG. Additional case studies are needed to determine to what extent our results can be truly generalized.

To that end, we have conducted initial experiments in a different industrial development project, this time in the electronics domain. In this case study, the meta-model is more complex, and the total set of documents is larger. Yet we can easily see the counterparts for the requirements and their categories, as well as the test cases and their categories. The initial results of these case studies yield traceability matrices and requirements views that are comparable in quality to the results from the LogicaCMG case.

Threats to Validity. We conclude our discussion with a brief analysis of potential threats to validity of the case study findings, conforming to [59].

A first concern is construct validity, which deals with the question whether the type of observations made can actually help in answering the case study's questions. The risks of subjective observations has been eliminated by the use of the REQANALYST tool suite, which automatically produces the data in the tables as discussed in Section 6. Obtaining an accurate reference matrix is perhaps the most subjective element of the case study, since the tool findings resulted in discussion on the correctness of the reference matrices produced. The process to carefully obtain this matrix was described in Section 6. A final issue related to construct validity is whether "validation percentage" is a reasonable measure for effort (reduction) – this question was discussed earlier in this section as well. In all cases, we actively involved various people from LogicaCMG in the case study, in order to minimize the risk of bias and subjective findings.

Since our case study is exploratory in nature, there are no threats to internal validity. With respect to external validity, we refer to the observations made above in the discussion on generalizing our findings.

Last but not least, repeatability ("reliability" in terms of [59]) is affected by the closed nature of an industrial case study like ours. Thus, while all data have been carefully collected and are indeed available, full repeatability is only possible within LogicaCMG. This is, in fact, important for LogicaCMG as well, since they are interested in conducting more studies like this one.

Revisiting the Case Study Questions. Before performing the case study, we had a few anticipations and expectations towards the likely outcome of a project like the one described here. One question was not initially related to the actual case study, and more of a general nature: which requirements views are needed in practice? According to the answers obtained from industrial partners, we concentrated on coverage views, lifecycle views, and status views. In particular, the first group, coverage views, is gaining importance, in many software domains, simply through the fact, that engineers want to assess the likely effect of a change in requirements on all other work products. For some domains, the automotive domain, for example, it will be compulsory in the future to provide such traceability views for certification.

Other questions, more fundamental to the case study performed, dealt with the how and the extent to which the traceability views in a system can be reverse-engineered from the existing work products. We have demonstrated the "how" sufficiently through application of LSI in our proposed MAREV method and its associated tool REQANALYST. LSI is capable to generate traceability links between documents that share the same inherent semantic concepts. It is quite robust with respect to the type and structure of the documents provided. Our case study is, therefore, successful in demonstrating the application of our method and tool in such a reconstruction context.

The question of the extent to which REQANALYST can reconstruct links correctly cannot be answered sufficiently in a single case study. We have seen that LSI can reconstruct, sometimes more, sometimes less traceability links for the required views. This depends on the parameters used, leading to high recall and low precision, or low recall, with high precision. The actual question to be answered here is whether and to which

extent missed links or many false positives are acceptable, and that depends on the quality of the reference matrix provided. The reference matrix is typically provided by the developers of a system as a result of some tedious manual process, and in other projects, we have observed that, sometimes, developers cannot agree on the right links, or they simply forgot to define links. The extent to which requirements views are reconstructed correctly is therefore still an open question that must be answered empirically through a number of similar case studies with thorough verification of the reference matrix, and this leads us the next questions asked earlier in this paper: can the approach be used to reconstruct traceability views, and can the reconstructed views help in real software development. The first question we answer with a definitive yes, but engineers have to decide whether low quality of the outcome is a serious hindrance for its application. Industry is often quite pragmatic in the application of automated tools: any little tool support is better than nothing, and only looking at and assessing automatically generated views might be a lot easier than creating them from scratch. At least the tool is capable of generating the most obvious links for views that are easy to establish. However, an aftertaste remains. That is the number of missed links. At the moment, there is no way to identify missed links without visiting all links, i.e. if there is no reference matrix. Therefore, we cannot claim our method is useful for software engineers as it is, nor can we say how much effort it can save, if any. We do not have conclusive data on the effort of reconstructing views manually. What we can foresee as future research, however, is an extended iterative reconstruction method in which the recall is increased gradually, generating many false positives, which can be filtered through a comparison with false positive links dismissed earlier. This may lead to a more precise reconstruction for which only a few new links would have to be assessed per iteration. That way, the method could bootstrap its own reference matrix and extend that on the way.

Another improvement could come from including a feedback mechanism similar to the one described in [22].

8 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:

- The identification, through a questionnaire among practitioners, of three relevant requirements views: coverage views, life-cycle path views, and status views.
- An approach to reconstruct these requirements views from software development work products, supported by our REQANALYST tool suite;
- The application of our approach to an ongoing project at LogicaCMG, illustrating
  1. how the software development process steers the reconstruction process and determines the meta-model used;
  2. how the quality of the reconstructed traceability matrix can vary per link type;
  3. how the traceability matrices can be used to obtain requirements views.

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. Last but not least, as mentioned in the previous section, we are presently working on an industrial case in the area of consumer electronics. This case concerns a globally distributed software development environment and a product-line, making it a very complex environment to apply our method.

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.

Bibliography

[1] Ian Alexander. Towards automatic traceability in industrial practice. In *Proc. of the 1st Int. Workshop on Traceability*, pages 26–31, Edinburgh, UK, 2002.

[2] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002. ISSN 0098-5589.

[3] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 020139829X.

[4] Shawn Bohner and Robert Arnold, editors. *Software Change Impact Analysis*. IEEE Computer Society, 1996.

[5] 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. ISBN 0-7695-2425-7.

[6] Paul Clements, David Garlan, Len Bass, Judith Stafford, Robert Nord, James Ivers, and Reed Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002. ISBN 0201703726.

[7] Paul Clements, David Garlan, Reed Little, Robert Nord, and Judith Stafford. Documenting software architectures: views and beyond. In *Proceedings of the 25th International Conference on Software Engineering*, pages 740–741, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

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

[9] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

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

[11] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: View-driven software architecture reconstruction. In *Proceedings Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, pages 122–134. IEEE Computer Society Press, 2004.

[12] Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Commun. ACM*, 41(12):54–62, 1998. ISSN 0001-0782.

[13] Steve Easterbrook and Bashar Nuseibeh. Managing inconsistencies in an evolving specification. In *Proc. of the 2th IEEE Int. Symposium on Requirements Engineering*, pages 48–55. IEEE Computer Society, 1995. ISBN 0-8186-7017-7.

[14] Matthias Findeis and Ilona Pabst. Functional safety in the automotive industry, process and methods. VDA (Verband der Automobilindustrie) Winter meeting 2006, 2006.

[15] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *Int. Journal of Softw. Eng. and Know. Eng.*, 2(1):31–58, March 1992.

[16] William B. Frakes and Ricardo Baeza-Yates, editors. *Information retrieval: data structures and algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. ISBN 0-13-463837-9.

[17] Orlena Gotel and Anthony Finkelstein. An analysis of the requirements traceability problem. In *Proc. of the 1st IEEE Int. Conf. on Requirements Engineering*, pages 94–101, Colorado springs, April 1994.

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

[19] H.-G. Gross, M. Lormans, and J. Zhou. Reformulating component identification as document analysis problem. In B. Shishkov J. Filipe, M. Helfert, editor, *2nd Intl Conference on Software and Data Technologies*, pages 111–116. Insticc Press, July, 22–24 2007. ISBN 978-989-8111-10-8.

[20] H.-G. Gross, M. Lormans, and J. Zhou. Towards software component procurement automation with latent semantic analysis. *Electronic Notes in Theoretical Computer Science*, 189:51–68, July 2007. ISSN 1571-0661.

[21] David C. Hay. *Requirements Analysis: From Business Views to Architecture*. Prentice Hall PTR, 2003.

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

[23] C. Hofmeister, R. Nord, and D. Soni. *Applied Software Architecture*. Addison-Wesley, 1999.

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

[25] IEEE. IEEE recommended practice for architectural description of software intensive systems (IEEE-std-1471), 2000.

[26] IEEE. IEEE guide for developing system requirements specifications (IEEE-std-1233), 1998.

[27] IEEE. IEEE recommended practice for software requirements specifications (IEEE-std-830), 1998.

[28] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Softw.*, 12(6): 42–50, 1995. ISSN 0740-7459.

[29] Jun Lin, Chan Chou Lin, Jane Cleland-Huang, Raffaella Settimi, Joseph Amaya, Grace Bedford, Brian Berenbach, Oussama Ben Khadra, Chuan Duan, and Xuchang Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 356–357, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2555-5.

[30] Mikael Lindvall and Kristian Sandahl. Practical implications of traceability. *Softw. Pract. Exper.*, 26(10):1161–1180, 1996. ISSN 0038-0644.

[31] 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.

[32] 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.

[33] Marco Lormans and Arie van Deursen. Reconstructing requirements traceability in design and test using latent semantic indexing. *Journal of Software Maintenance and Evolution: Research and Practice*, 2008. Submitted. Preprint on line available as technical report TUD-SERG-2007-007. Delft University of Technology.

[34] 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 *Proceedings International Workshop on Principles of Software Evolution (IWPSE)*, pages 149–158. IEEE Computer Society, 2004.

[35] Marco Lormans, Hans-Gerhard Gross, Arie van Deursen, Rini van Solingen, and Andre Stéhouwer. Monitoring requirements coverage using reconstructed views: An industrial case study. In *Proc. of the 13th Working Conf. on Reverse Engineering*, pages 275–284, Benevento, Italy, October 2006. IEEE Computer Society.

[36] 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.

[37] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Can information retrieval techniques effectively support traceavbility link recovery? In *Proceedings 10th International Conference on Program Comprehension*, pages 307–316, Athens, Greece, 2006. IEEE Computer Society.

[38] Andrea De Lucia, Rocco Oliveto, Francesco Zurolo, and Massimiliano Di Penta. Improving comprehensibility of source code via traceability information: a controlled experiment. In *Proc. of the 14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 317–326, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2601-2.

[39] Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Francesco Zurolo. Coconut: Code comprehension nurturant using traceability. In *Proc. of the 22nd IEEE Int. Conf. on Software Maintenance (ICSM'06)*, pages 274–275, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2354-4.

[40] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodolology*, 16(4):Art. nr. 13, 2007.

[41] 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 2nd Int. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 47–54, Montreal, Canada, 2003. TEFSE 2003.

[42] Andrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proc. of the 25th Int. Conf. on Software Engineering*, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1877-X.

[43] Andrian Marcus, Xinrong Xie, and Denys Poshyvanyk. When and how to visualize traceability links. In J.I. Maletic, J. Cleland-Huang, J. Huffman Hayes, and G. Antoniol, editors, *3rd Intl Workshop on Traceability in Emerging Forms of Software Engineering*, pages 56–61, Long Beach, California, November 2005.

[44] 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. ISSN 0740-7459.

[45] 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. ISSN 0098-5589.

[46] 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. ISSN 0740-7459.

[47] S. Park, H. Kim, Y. Ko, and J Seo. Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. *Information and Software Technology*, 42(6):429–438, 2000.

[48] Massimiliano Di Penta, Sara Gradara, and Giuliano Antoniol. Traceability recovery in rad software systems. In *Proc. of the 10th Int. Workshop on Program Comprehension*, pages 207–216, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1495-2.

[49] 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. ISBN 0-8186-7017-7.

[50] Bala Ramesh and Matthias Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001. ISSN 0098-5589.

[51] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979. ISBN 0408709294.

[52] James Robertson and Suzanne Robertson. Volere requirements specification template. Technical report, Atlantic Systems Guild, 2000.

[53] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

[54] Raffaella Settimi, Jane Cleland-Huang, Oussama Ben Khadra, Jigar Mody, Wiktor Lukasik, and Chris DePalma. Supporting software evolution through dynamically retrieving traces to UML artifacts. In *Proc of the 7th Int. Workshop on Principles of Software Evolution*, pages 49–54, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2211-4.

[55] 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. ISBN 3-540-66954-X.

[56] Arie van Deursen and Leon Moonen. Documenting software systems using types. *Science of Computer Programming*, 60(2):205–220, April 2006.

[57] 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. ISBN 1-58113-508-4.

[58] 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. ISBN 0-7695-1465-0.

[59] R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications Inc, 3d edition, 2003.

[60] John A. Zachman. A framework for information systems architecture. *IBM Syst. J.*, 26(3):276–292, 1987. ISSN 0018-8670.

[61] Andrea Zisman, George Spanoudakis, Elena Perez-Mi nana, and Paul Krause. Tracing software requirements artifacts. In *Proc. of Int. Conf. on Software Engineering Research and Practice*, pages 448–455, Las Vegas, Nevada, USA, 2003.

[62] Xuchang Zou, Raffaella Settimi, Jane Cleland-Huang, and Chuan Duan. Thresholding strategy in requirements trace retrieval. In *CTI Research Symposium*, pages 100–103, Chicago, 2004.

TUD-SERG-2008-032
ISSN 1872-5392