Delft University of Technology
Software Engineering Research Group
Technical Report Series

Reconstructing Requirements Traceability in Design and Test Using Latent Semantic Indexing

Marco Lormans and Arie van Deursen

Report TUD-SERG-2007-007

TUD-SERG-2007-007

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/

Reconstructing Requirements Traceability in Design and Test using Latent Semantic Indexing[§]

Marco Lormans[1,*,†], Arie van Deursen[1,2,‡]

[1] *Delft University of Technology, Delft, The Netherlands*
[2] *CWI, Amsterdam, The Netherlands*

Abstract

Managing traceability data is an important aspect of the software development process. In this paper we define a methodology, consisting of six steps, for reconstructing requirements views using traceability data. One of the steps concerns the reconstruction of the traceability data. We investigate to what extent Latent Semantic Indexing (LSI), an information retrieval technique, can help recovering the information needed for automatically reconstructing traceability of requirements during the development process. We experiment with different link selection strategies and apply LSI in multiple case studies varying in size and context. We discuss the results of a small lab study, a larger case study and a large industrial case study.

KEY WORDS: Requirements Traceability; Traceability Reconstruction; Information Retrieval

1 Introduction

For many organisations, the purpose of requirements management is to provide and maintain clear agreements between the different stakeholders involved in developing a product, while still allowing requirements evolution. Requirements management is a supportive process that starts in the orientation phase and continues during the rest of the product development life-cycle.

Managing requirements in such a way that useful information can be extracted from this requirements set, is hard in practice [20, 23]. This extracted information can be used for many applications such as generating requirements views or impact analysis [9]. The requirements views we will encounter are coverage views, which include whether or not a requirement is covered by an acceptance test, by a design artifact, by a system test, and so on. These requirement views can provide a major asset for developers and project managers, offering them a way to monitor the progress of the development process.

Obtaining accurate information requires that an up-to-date traceability matrix is maintained, establishing links between, for example, requirements and test cases. Keeping the traceability links consistent during development of the product is a time consuming, error-prone, and labor-intensive

[*]Correspondence to: Marco Lormans, Software Technology Department, Delft University of Technology, P.O. box 5031, NL-2600GA, Delft, The Netherlands.
[§]This is a substantially revised and expanded version of our papers [31] and [32]
[†]E-mail: M.Lormans@tudelft.nl
[‡]E-mail: Arie.vanDeursen@tudelft.nl

process demanding disciplined developers [7, 19, 33]. Currently available tools do not support the feature of automatically recovering traceability links [2, 20, 23].

In this paper, we investigate to what extent relevant traceability links can be reconstructed automatically from available documents using latent semantic indexing (LSI). LSI is an information retrieval method assuming there is a latent semantic structure for every document set [13]. LSI creates a "semantic" subspace of terms and documents closely associated using statistical techniques. This subspace can be used for retrieving information and, in our case, for reconstructing traceability links. For this reason, our approach assumes a document-oriented requirements engineering process based on natural language, which can identify semantic similarities between different documents produced during the development of the product.

The long term objective of our research is to determine how industry can benefit from using LSI to track and trace requirements and eventually generate various requirements views. In this paper, we describe three exploratory case studies, that give answers to the following questions:

1. Can LSI help in reconstructing meaningful traceabilily relations between requirements and design, and between requirements and test cases?

2. What is the most suitable strategy for mapping LSI document similarities to reconstructed links?

3. What are the most important open issues that need to be resolved before LSI can be applied successfully in an industrial context?

To answer these questions we offer, in every case study, an analysis why the particular links can be reconstructed and what the requirements are for documenting the related development work products.

The three case studies in which we applied LSI, vary in size and context. The first is a lab study, Pacman, used in a testing course at Delft University of Technology. Available documentation includes use cases, design decisions, acceptance test cases, as well as a Java implementation with a JUnit test suite. The Pacman case gives us the opportunity to explore all the possibilities of the techniques in a controlled environment. In this study, we varied the different parameters of our analysis to come to a setting giving the best results. The second case study is part of a software engineering course at Eindhoven University of Technology. In this course, a group of students need to develop a complete new software system from scratch and properly document requirements, design and test cases including tracebility links. The last case study is an industrial case study carried out at Philips Applied Technology. In this study, requirements, design decisions, and corresponding test suite for a Philips DVD recorder were analyzed.

The remainder of this paper is organized as follows. In Section 2 we give an overview of background information and discuss related work, followed by a brief survey of latent semantic indexing in Section 3. In Section 4 we describe our link reconstruction methodology, MAREV, and in Section 5 we describe the link selection strategies we use in this methodology. Next, in Section 6 we describe the tool we developed to support our approach. The three cases studies are presented in Section 7. In Section 8 we compare and discuss the results of the case studies. We conclude the paper by summarizing the key contributions and offering suggestions for future research.

2 Background and Related Work

2.1 Requirements Views

The different perspectives on requirements are often represented using views. Views capture a subset of the whole system in order to reduce the complexity from a certain perspective. For example, Nuseibeh *et al.* discuss the relationships between multiple views of a requirements specification [41]. This work is based on the viewpoints framework presented by Finkelstein *et al.* in [17]. This framework primarily helps organizing and facilitating the viewpoints of different stakeholders.

Von Knethen also discusses view partitioning, but from a different perspective [53]. She considers views on the system, distinguishing, e.g., the static structure from the dynamic interactions in the system.

If we look beyond requirements, the concept of "view" also appears in the area of 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.* [24], the IEEE standard 1471 [26], and the views discusses by Clements *et al.* in their book "Documenting Software Architetures" [11, 12]. Finally, Van Deursen *et al.* discuss a number of specific views for architecture reconstruction [14].

Although much research is done in the area of (requirements) views, there is no general agreement on what these views should look like or what information they should contain. Every project setting seems to have its own specific information needs concerning requirements.

2.2 Requirements Traceability and Reference Models

Managing different requirements perspectives (views) can be supported through appropriate meta-models, as shown by Nissen *et al.* [40]. An important area of research in the domain of traceability is developing these meta-models. These so called reference models discussed in [37, 46, 52, 53, 55] define the development artifacts including their attributes, and the traceability relations that are allowed to be set between these artifacts.

Von Knethen proposes (conceptual) traceability models for managing changes on embedded systems [53, 54]. These models help estimating the impact of a change to the system or help to determine the links necessary for correct reuse of requirements. According to Von Knethen, defining a workable traceability model is a neglected activity in many approaches. Our earlier research confirms the importance of defining a traceability model [33]. The initial experiments concerned a static traceability model. New insights suggest a dynamic model, in which new types of links can be added as the way of working evolves during the project. The need for information as well as the level of detail changes [16].

2.3 Traceability Reconstruction

To reconstruct coverage views from project documentation we need some traceability support. Several traceability recovery techniques already exist, each covering different traceability issues during the

development life-cycle. Some discuss the relations between source code and documentation, others the relations between requirements on different levels of abstraction.

Antoniol *et al.* use information retrieval (IR) methods to recover the traceability relations from C++ code onto manual pages and from Java code to requirements [3]. Marcus and Maletic, and Di Penta *et al.* use latent semantic indexing for recovering the traceability relations between source code and documentation [38, 39, 45]. The IR methods in these cases are mostly applied to reverse engineering traceability links between source code and documentation in legacy systems.

IR methods can also be used for recovering traceability links between the requirements themselves [21, 42]. In these cases, traceability recovery is mainly used for managing the requirements after development when all the documentation needs to be finalized and released. Both Natt och Dag *et al.* and Huffman Hayes *et al.* have developed a tool to support their approach. In [43] Natt och Dag *et al.* discuss their approach and tool, called ReqSimile, in which they have implemented the basic vector space model and applied it in an industrial case study. Huffman Hayes *et al.* have implemented various methods for recovering the traceability links in their tool called RETRO [22, 23]. They also applied their approach in an industrial case study.

De Lucia *et al.* present an artifact management system, which has been extended with traceability recovery features [34, 35]. This system manages all different artifacts produced during development such as requirements, designs, test cases, and source code modules. De Lucia *et al.* also use LSI for recovering the traceability links. In [36], they improved their traceability recovery process and propose an incremental approach. In this approach they incrementally try to identify the "optimal" threshold for recovering traceability links.

Cleland-Huang *et al.* define three strategies for improving the dynamic requirements traceability performance: hierarchical modeling, logical clustering of artifacts and semi-automated pruning of the probabilistic network [10]. They are implementing their approach in a tool called Poirot [30]. Furthermore, like De Lucia *et al.*, they have defined a strategy for discovering the optimal thresholds [56].

Finally, IR techniques are also used for improving the quality of the requirements set. Park *et al.* use the calculated similarity measures for improving the quality of the requirements specifications [44].

3 Latent Semantic Indexing

Latent Semantic Indexing (LSI) is an information retrieval technique based on the vector space model and assumes that there is an underlying or latent structure in word usage for every document set [13]. This is particularly caused by classical IR issues as *synonymy* and *polysemy*. Synonymy concerns the fact that there are many ways to refer to the same object. Users in different contexts, or with different needs, knowledge, or linguistic habits will describe the same information using different terms. Polysemy involves the fact that most words have more than one distinct meaning. In different contexts or when used by different people the same term takes on varying referential significance [13].

LSI uses statistical techniques to estimate the latent structure of a set of documents. A description of terms and documents based on the underlying latent semantic structure is used for representing and retrieving information. This way LSI partially overcomes some of the deficiencies of assuming independence of words, and provides a way of dealing with synonymy automatically.

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 [5, 50]. Any rectangular matrix, for example a $t \times d$ matrix of terms and documents, X, can be decomposed into the product of three other matrices:

$$X = T_0 S_0 D_0^T$$

such that T_0 and D_0 have orthonormal columns and S_0 is diagonal (and D_0^T is the transpose of D_0). This is called the singular value decomposition of X. T_0 and D_0 are the matrices of left and right singular vectors and S_0 is the diagonal matrix of singular values.

SVD allows a simple strategy for optimal approximate fit using smaller matrices. If the singular values in S_0 are ordered by size, the first k largest values may be kept and the remaining smaller ones set to zero. The product of the resulting matrices is a matrix X' which is only approximately equal to X, and is of rank k. Since zeros were introduced into S_0, the representation can be simplified by deleting the zero rows and columns of S_0 to obtain a new diagonal matrix S, and deleting the corresponding columns of T_0 and D_0 to obtain T and D respectively. The result is a reduced model:

$$X' = TSD^T \approx X$$

which is the rank-k model with the best possible least square fit to X [13].

Note that the choice of k is critical: ideally, we want a value of k that is large enough to fit all the real structure in the data, but small enough so we do not also fit the sampling error or unimportant details. Choosing k properly is still an open issue in the factor analytic literature [13]. Our choice will be discussed when we apply LSI in our case studies.

Once all documents have been represented in the LSI subspace, we can compute the similarities between the documents. We take the cosine between their corresponding vector representations for calculating this similarity metric. This metric has a value between [-1, 1]. A value of 1 indicates that two documents are (almost) identical.

These measures can be used to cluster similar documents, or for identifying traceability links between the documents. We can also define new queries and map these into the LSI subspace. In this case, we can identify which existing documents are relevant to the query. This can be useful for identifying requirements in the existing document set.

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 or manipulation of the input, which drastically reduces the costs of traceability link recovery [34, 37]. However, some text transformations are needed to prepare the documentation to form the corpus of LSI. This user-created corpus will be used as the input for creating the term-by-document matrix.

4 MAREV: A Methodology for Automating Requirements Evolution using Views

The long term objective of our work is an approach that supports large organizations in the software industry in managing requirements throughout the life-cycle of, for example, consumer electronics products or document systems such as copiers. Such products need to fulfil hundreds or thousands of

requirements. Furthermore, these requirements can change over time when new product versions are created and shipped.

Our focus is on reconstructing *requirements views*, i.e., views on the set of requirements that can be used to monitor the progress in requirements development, design, and testing. In this paper we focus on the reconstruction of requirements traceability needed to generate the requirements views.

In order to answer the questions raised in the introduction, we conducted three case studies, which are described in Section 7. In this section we discuss 1) the steps of our methodology MAREV, to reconstruct the traceability links and generate requirements views, and 2) the approach we used to assess the reconstructed links. In Section 5 we describe the link selection strategies (step 5) in more detail and in Section 6 we discuss the tool that we developed in order to carry out these steps.

4.1 Link Reconstruction Steps

We have developed an approach for reconstructing our requirements views automatically. In this particular case we experimented with LSI for reconstructing the traceability links (step 4), which resulted in reasonably good traceability recovery results. The steps are:

1. Defining the underlying traceability model;
2. Identifying the concepts from the traceability model in the available set of documents;
3. Preprocessing the documents for automated analysis;
4. Reconstructing the traceability links;
5. Selecting the relevant links;
6. Generating requirements views.

In this paper, we will primarily focus on techniques for executing step 4 and 5 handling the traceability recovery and selection of correct links. Of course, step 1, 2 and 3 are of major importance for executing step 4 and 5 successfully. We have defined some requirements views for step 6, but this remains future work for now. We will discuss all steps briefly and then focus on the steps 4 and 5 in the case studies.

4.1.1 Traceability Model Definition

Traceability relations establish links between requirements on the one hand and various types of development work products on the other. A traceability model defines the work products and the types of links that are permitted within the development process.

The choice of traceability model mainly depends on the purposes for which it is to be used. For example, Ramesh and Jarke [46] discuss a range of different traceability models. Other examples of reference models can be found in [37, 52, 53, 55].

An example of a traceability model relevant for coverage monitoring is shown in Figure 1. This model and the work products, including their dependencies, contained in it reflect the way of working at a big industrial company, Philips Applied Technologies, in the embedded systems domain.

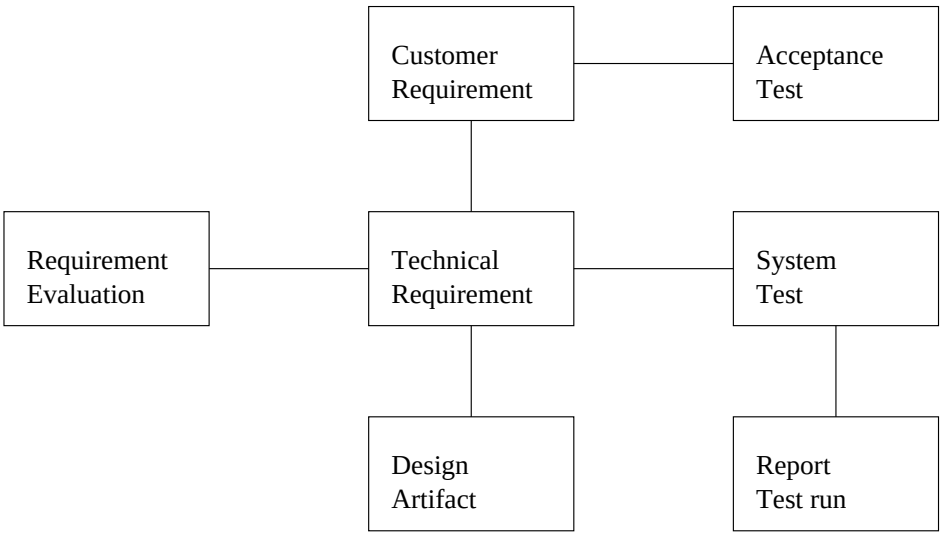


Figure 1: Traceability Model

For example, it distinguishes between a *customer requirement* (cast in terms familiar by customer representatives) and *technical requirements* (cast in terms familiar by developers). Moreover, the model supports *evaluation* of requirements: after shipping the product, field studies are conducted in order to evaluate the working of the requirement in real life. The evaluation results are taken into account when shipping a new version of the product. This traceability model enables us to derive, for example, the following coverage information that can be included in a requirements view:

- **Identification coverage**; The information in this view indicates the links between customer requirements and technical requirements. The technical requirements specify the product that actually will be built. Every system requirement should have a link with at least one customer requirement, and vice versa.
- **Design coverage**; Design coverage captures the information to ensure that the requirements in the system's requirements specification are addressed in the design. This view shows how well the design reflects the requirements. Note that the presence of a link does not mean that these requirements are correctly designed or implemented. Having a requirements coverage of 100% after the design phase tells management that the system should have all functionality covered in the design as agreed in the contract.
- **Test case coverage**; A comparable situation applies to the requirements coverage in the test specifications. Most test specifications are created in the design phase in parallel with the design. This view shows how well the test specification reflects the requirements. Again this does not mean the functionality is correctly implemented. Having a coverage of 100% tells management that all functionality will be tested in the test phase.

- **Test pass coverage**; A system test is an internal test, often called factory test, to check if the system is working correctly. If all system tests pass, the development team can show the customer that all functionality is implemented and that all functionality is working correctly as agreed. This view shows which requirements are tested and ready for the customer acceptance test.
- **Acceptance coverage**; The customer can approve the results by means of the final acceptance test. This view shows which requirements are accepted by the customer and are ready for release.
- **Evaluation coverage**; After delivery, the evaluation coverage view indicates which requirements have been evaluated and are suitable for reuse in ongoing and future projects.

The above discussed examples of coverage views each reflect a link in the traceability model depicted in Figure 1. Of course other examples can be defined such as combinations of these views, e.g., capturing information about the coverage in the test specification and the actual execution of these system tests.

The work products and traceability links that actually need to be captured in the traceability model depend on project-specific information needs [16], but also on factors such as schedule and budget [47].

4.1.2 Concept Identification

Every concept contained in the traceability model should be uniquely identified in the available documentation. Since the documents are typically semi-structured (typically being just MS Word files), this requires a certain amount of manual processing. The more systematically the documents are organized (for example through the use of templates such as MIL-std 498 or IEEE-1233-1998), the easier it is to make this structure explicit and identify the texts for each of the entities from the traceability model.

In general, identifying individual requirements and test cases in the documentation is relatively easy compared to identifying the design artifacts. Requirements and test cases in most development approaches are tagged with a unique identifier. For design decisions it is often not so clear how they should be documented and identified. Key decisions are often captured in diagrams, e.g., in UML. Here, we encounter the well known problem of establishing traceability relations between requirements and design [51]. Solutions exist to make architectural knowledge more explicit such as capturing architectural assumptions explicitly [29]. Unfortunately these solutions are often not yet used in practice [20].

4.1.3 Text Preprocessing

After defining the entities and the texts belonging to each of them, some pre-processing of these texts is needed. The first step is extracting the texts from the original documents, bringing them in the (plain text) input format suitable for further automated processing. This often is a manual or semi-automatic task. In the semi-automatic case, scripting techniques using, e.g., Perl, can be used to transform the original text into the format needed for further processing. Whether such scripting techniques can be used depends very much on the document structure of the original documentation. The next step is to conduct typical IR steps such as lexical analysis, stop word elimination, stemming, index-term selection, and index construction.

	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	0,96	0,87	0,89	0,77	0,85	0,79	0,89	0,96	0,81	0,95
D2	0,89	0,79	0,83	0,79	0,81	0,8	0,85	0,89	0,83	0,91
D2.1	0,81	0,73	0,76	0,59	0,69	0,65	0,77	0,87	0,69	0,82
D2.2	0,74	0,83	0,88	0,9	0,8	0,9	0,81	0,66	0,79	0,73
D2.3	0,84	0,96	0,85	0,72	0,98	0,77	0,84	0,81	0,77	0,83
D2.4	0,95	0,86	0,95	0,81	0,81	0,85	0,96	0,94	0,94	0,96
D2.5	0,96	0,88	0,89	0,77	0,86	0,83	0,94	0,94	0,9	0,96
D3	0,96	0,89	0,89	0,77	0,89	0,82	0,92	0,93	0,86	0,95
D3.0	0,91	0,85	0,91	0,88	0,81	0,92	0,97	0,84	0,98	0,91
D3.1	0,97	0,87	0,88	0,78	0,85	0,83	0,94	0,93	0,87	0,97
D3.2	0,91	0,96	0,91	0,87	0,93	0,91	0,96	0,84	0,92	0,88
D3.2.1	0,86	0,95	0,91	0,8	0,95	0,85	0,9	0,8	0,85	0,86
D3.2.2	0,88	0,95	0,95	0,87	0,91	0,91	0,94	0,81	0,92	0,87
D3.2.3	0,79	0,92	0,82	0,86	0,92	0,89	0,87	0,7	0,84	0,76
D3.3	0,83	0,92	0,83	0,88	0,93	0,91	0,89	0,73	0,86	0,8
D3.4	0,98	0,87	0,9	0,8	0,85	0,85	0,96	0,94	0,91	0,98
D3.5	0,9	0,84	0,92	0,9	0,77	0,93	0,97	0,83	0,99	0,89

Figure 2: Example of Similarity Matrix

The collection of documents (the "corpus") to be used as input for LSI may be larger than the texts corresponding to the entities from the traceability model. In fact, LSI analysis may benefit from including additional documents containing texts about, for example, the application domain. It allows LSI to collect more terms that are typically used in combination, helping LSI to deal with, for example, synonyms. If such extra documents are used, these documents need to be preprocessed as well.

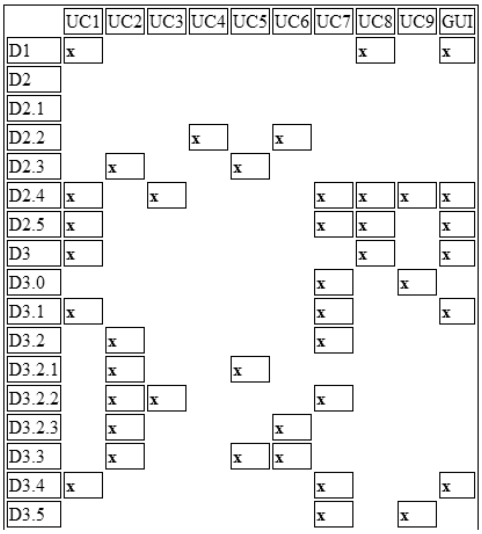
4.1.4 Link Reconstruction

After generating the term-by-document matrix we can reconstruct the traceability links using LSI. First of all, this creates the rank-*k* model on the basis of which similarities between documents can be determined. Here we need to choose the number for *k*. Secondly, for every link type in the traceability model (for example tracing requirements to designs) a similarity matrix is created containing the similarities between all elements (for example between every requirement and design artifact).

The result of our LSI analysis is a similarity matrix containing the recovered links, represented as their similarity measures. Figure 2 shows an example of a similarity matrix calculated for 10 use cases and 3 design components. This similarity matrix allows us to judge the quality for every recovered link.

4.1.5 Link Selection

Once LSI has created the similarity matrix, a choice has to be made if the similarity number is indeed a traceability link or not. There are several different strategies for doing this: the cut point, cut percentage, constant threshold, and variable threshold strategy [3, 34, 39].



	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	GUI
D1	x							x		x
D2										
D2.1										
D2.2				x		x				
D2.3		x			x					
D2.4	x		x				x	x	x	x
D2.5	x						x	x		x
D3	x							x		x
D3.0							x		x	
D3.1	x						x			x
D3.2		x					x			
D3.2.1		x			x					
D3.2.2		x	x				x			
D3.2.3		x				x				
D3.3		x			x	x				
D3.4	x						x			x
D3.5							x		x	

Figure 3: Example of Traceability Matrix

All these strategies have their strenghts and shortcomings. In order to benefit from the specific characteristics of applying LSI to traceability reconstruction, we propose two new link selction strategies: a one and two dimensional vector filter stragety, which we will discuss in detail in Section 5.

The result of the link selection step is a traceability matrix containing the links not filtered by the chosen link selection strategy. The link selection strategy and its corresponding parameters determine which similarity measures will become the final traceability links. In Figure 3, an example of a reconstructed traceability matrix is shown using our two dimensional vector filter strategy. In Section 5, we will explain, using an example, how to construct this traceability matrix with our link selection strategies.

4.1.6 Requirements View Generation

The final step is to use the reconstructed traceability links to obtain requirements views. Currently we have defined a number of different views concerning the status and coverage of requirements, as well as a view to browse the reconstructed traceability links.

For example, given the presence of a link, the status of a requirement can be appropriately set. Moreover, management information can be obtained by computing percentages of requirements that have reached a certain status.

The reconstructed traceability matrix can also be used to calculate coverage metrics. Currently, for every link defined in the traceability model we calculate the percentage of all requirements covered by the specific link. Thus, we get a list of requirements coverage percentages in the design, test cases, and so on. Another view shows the requirements that are not covered by, e.g., the design or test cases.

The developers can use this information to check if the requirements are indeed not covered and can undertake appropriate action.

4.2 Assessment of the Results

In order to assess the suitability of the reconstructed links, we conduct a qualitative as well as a quantative analysis of the links obtained.

The qualitative assessment of the links is primarily done by experts exploring the documents. The structure of the documents set is of major influence on this process. It helps significantly if the documents are structured according to an (international) standard or template such as IEEE standard 830-1998, IEEE standard 1233-1998, ESA [1] or Volere [49]. Beforehand, such a structure helps choosing the concepts and preprocessing the documents. Afterwards it helps in assessing the reconstructed traceability links as it is easier to browse through the documents. A tool for exploring the links in order to support the qualitative assessment is discussed in Section 6.

The quantitative assessment consists of measuring two well-known IR metrics: *recall* and *precision* [4, 18, 48, 50]:

$$recall = \frac{|correct \cap retrieved|}{|correct|}$$

$$precision = \frac{|correct \cap retrieved|}{|retrieved|}$$

The number of *correct* traceability links are specified in a reference traceability matrix provided by the experts developing the system. The number of *retrieved* traceability links is derived from the LSI analysis.

Both metrics have values between [0, 1]. A recall of 1 means that all correct links were reconstructed, however the total set of links can contain incorrect links. A precision of 1 indicates that all reconstructed links are correct, but there can be correct links that were not reconstructed. The link selection strategy and its corresponding parameters influence the performance indicators recall and precision, as we will see in the case studies.

5 Link Selection Strategies

For selecting the relevant links in a similarity matrix several link selection strategies are available. In their application of LSI, De Lucia *et al.* present a number of strategies for selecting traceability links. The following are discussed [34]:

1. *cut point*; In this strategy we select the top μ links regardless of the actual value of the similarity measure [3, 39]. This strategy always returns exactly μ traceability links.

2. *cut percentage*; In this strategy we select a percentage p of the ranked list to be considered as links regardless of the actual value of the similarity measure. This strategy always returns exactly the p% of the total reconstructed candidate links.

3. *constant threshold*; In this strategy we select those links that have a similarity measure higher than *c*, where *c* is a constant (a commonly used threshold is 0.7). Note that the number of returned links is flexible.

4. *variable threshold*; In this strategy, proposed by De Lucia *et al.*, we select those links that have a similarity measure higher than ε, where ε is calculated through a percentage *q* of the difference between the maximum and minimum similarity measures of the total set of similarity measures, e.g., the best *q*% of the interval defined by the maximum and minimum [34]. This strategy is useful if the difference between the maximum and the minimum is low.

5. *scale threshold*; In this strategy, proposed by De Lucia *et al.*, the links are obtained as a percentage of the maximum similarity measures, i.e., $\varepsilon = c * MaxSimilarity$, where $0 \leq c \leq 1$ [3, 34]. This measure is most useful if the maximum similarity is low.

In our case studies, we have experimented with each of these strategies, which again all have their strengths and shortcomings. Except for strategy constant threshold, all strategies return at least one or more traceability links as correct links, while in our case studies situations exist where no links should be found, e.g., when the quality of the document set is poor. Note however, that it is possible for individual rows or columns to have no links, since the threshold is calculated using the complete set of similarity measures in the matrix.

Furthermore, the first two strategies do not take the similarity measure into account and make a selection independent of the calculated result. They simply select the μ best or *p*% best similarity measures as traceability links. A typical question is what number should we choose for the μ and *p*? In most cases, we do not know the exact number of traceability links to return and it is hard to predict this number.

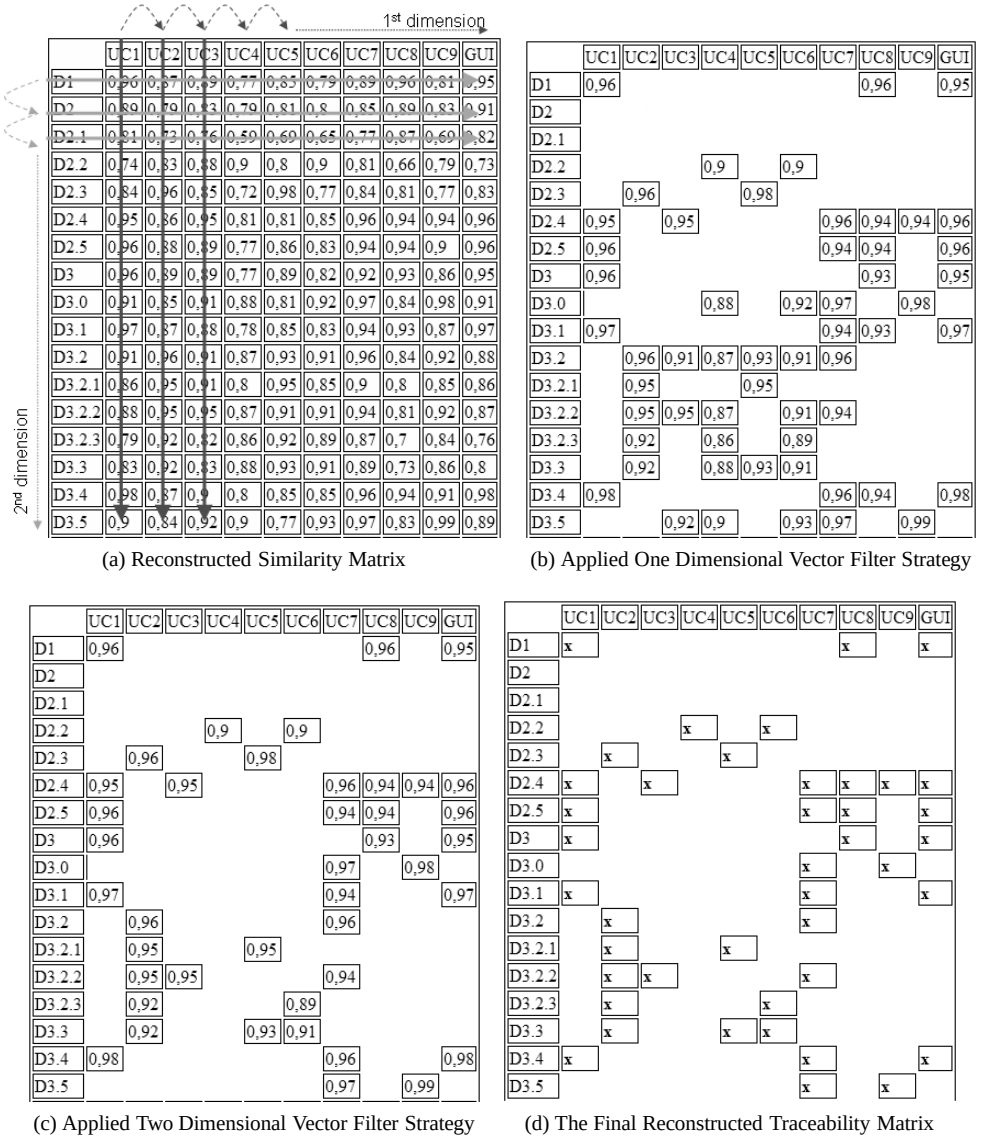
The last two strategies define an interval containing the selection of similarity measures that are correct traceability links. Both strategies are very vulnerable for extremes. For example, if the minimal similarity measure is very low with respect to the other measures, it is possible that the top 20% contains almost all measures.

To deal with these issues, we have experimented with a new approach, that tries to take advantage of the specific characteristics of our setting. For requirements traceability purposes, it is not very likely that there are, e.g., requirements that link to all test cases, or design decisions that may be inspired by all requirements together. For that reason, we propose a strategy that works on a per column basis.

5.1 One Dimensional Vector Filter Strategy

This strategy takes into account each column of the similarity matrix separately (see 1st dimension in Figure 4a). Each column vector of the similarity matrix is taken as a new set of similarity measures. Then, for each column, it combines the constant and variable threshold approaches: if there are measures above the constant threshold *c*, we take the best *q*%, e.g., 20% of the similarity measures in that column.

The constant threshold is used to indicate if there is any similarity between this specific work product (in the example a use case) and the other work products (in the example the design artifacts)(see Figure 4a). If all similarity measures in the column vector are smaller than *c*, there is not enough



(a) Reconstructed Similarity Matrix

(b) Applied One Dimensional Vector Filter Strategy

(c) Applied Two Dimensional Vector Filter Strategy

(d) The Final Reconstructed Traceability Matrix

Figure 4: Applying the One and Two Dimensional Vector Filter on the example Similarity Matrix using $c = 0.7$ and $q = 20\%$

similarity between the work products and thus there are no traceability links at all. This way we can guarantee a certain level of quality for our reconstructed traceability links.

If there are measures greater than the constant threshold we take the variable threshold for selecting the traceability links. With the variable threshold, a similarity interval is defined by the minimum and maximum similarity measures of the column vector (taking the original column vector, including the possible measures smaller than c). We use q to calculate the variable threshold ε per column. This threshold ε retains the best q% of the similarity measures in that vector representation and selects them as traceability links independent of the other column vectors.

In the example, depicted in Figure 2 and Figure 4a, we can see that the constant threshold ($c = 0.7$) has no influence on the result. All the similarity measures and higher than $c = 0.7$. Consequently, every use case in this example has at least one link (in this case, the variable threshold always returns a link per column).

The variable threshold ε is calculated per column, and differs for each column; UC1 $\Rightarrow \varepsilon = 0.916$, UC2 $\Rightarrow \varepsilon = 0.914$, UC3 $\Rightarrow \varepsilon = 0.912$, UC4 $\Rightarrow \varepsilon = 0.838$,UC5 $\Rightarrow \varepsilon = 0.922$, etc... We see that the relatively high variable threshold of UC5 would cause UC4 not to return any links, and that the relative low variable threshold of UC4 would cause that only 2 links of UC2 are filtered (see Figure 4a and Figure 4b). Besides that, every column can have a different number of returned links. Note that the standard variable threshold strategy would use a single ε for all columns.

With respect to applying only the variable threshold strategy, we will see in our case studies that our strategy increases the precision of the result without affecting the recall. The variable threshold ε in case of taking $q = 20\%$ results in $\varepsilon = 0.91$ for all columns. Consider, as an example, the effect of this threshold on UC4. With this threshold, UC4 would have no links, while for the given example we know there should be two links returned. This decreases the recall with respect to using our one dimensional vector filter strategy. On the other hand, our strategy does filter more of the relative high similarity measures of UC9. Our strategy with a $\varepsilon = 0.930$ for UC9 returns only three links, while with the "normal" variable threshold ($\varepsilon = 0.91$) it returned five links. As we will see in the case studies, the correct link is indeed in that set of three links.

The benefits of our one dimensional vector filter strategy, compared to the strategies discussed by De Lucia *et al.* [34], are the following:

- Our strategy is flexible in the number of returned candidate traceability links. Thus, it does not always return an absolute number of links, like the cut point and cut percentage strategies.
- Our strategy takes into account the calculated similarity measures and uses a constant threshold to guarantee a certain level of quality. It is possible that with our strategy no traceability links are returned.
- Our strategy is less vulnerable for extremes in the total set of similarity measures. It only takes a subset (the column vector) to set the variable threshold. For each individual work product, it returns a more precise set of traceability links that is not affected by the similarity measures in the other column vectors.

We have shown some arguments why our strategy improves the link selection step compared to the other available strategies. However, there are still two problems with this strategy: 1) it does not

consider the other dimension (row) of the similarity matrix (in our case example the design vectors) and 2) it always returns a link for each column if the constant threshold is too low.

The first is a problem because of the following. Imagine the situation that a design vector has relatively high values for the similarity measures compared to the other design vectors in the matrix, e.g., D3.2 compared to D2.2. In this case, this design artifact returns many traceability links using our one dimensional vector filter strategy; the similarity measures are higher than the constant threshold c and also belong to the interval defined by ε of each column. This is an undesirable situation as one design artifact (or one test case) should not cover all (or most of the) use cases.

The second is a problem because it should be possible for columns to return no links. For example, when a use case is not yet covered in the design, the column of that use case should not return any links. Both problems are solved using our second strategy, which is an extension to the one dimensional vector filter strategy.

5.2 Two Dimensional Vector Filter Strategy

This two dimensional vector filter strategy is basically the same as our one dimensional vector filter strategy except that it is executed on both dimensions of the similarity matrix (see Figure 4a). It also filters the relatively weak similarity measures of the row (in our example the design vectors). In general, this should improve the quality of the reconstructed traceability links; the precision further increases.

When applying our two dimensional vector filter strategy in our example we see that for D3.2, four extra links are filtered. The same results can be observed for, e.g., D3.2.2 and D3.5 (see Figure 4c).

However, with this second strategy the risk increases that the filter is too precise and also eliminates correct links, thus decreasing recall. If we look again at UC4, we see there only remains one link after applying our two dimensional vector filter strategy. After applying the two dimensional vector filter strategy, we transform the remaining similarity measures to traceability links. Finally, these form the traceabilty matrix depicted in Figure 4d.

The additional benefits of our two dimensional vector filter strategy with respect to the benefits discussed in Section 5.1 are the following:

- It returns a more precise result for each pair of work products (in our example; use case - design artifact pairs).
- It possible that a column returns no links even if the constant threshold has no influence on the result. The second filter dimension (per row) makes this possible.

6 The ReqAnalyst Tool Suite

In order to support the traceability reconstruction approach, we developed a tool called ReqAnalyst[†]. The objectives of this tool are:

[†]This tool suite replaces the tool support (TMG toolbox, Trace Reconstructor and Trace Explorer) used in our paper at CSMR 2006 [32]. The tool is available from `http://swerl.tudelft.nl/bin/view/Main/ReqAnalyst`.

- To offer a test bed for experimenting with different traceability reconstruction approaches and algorithms;
- To support the application of these approaches to industrial projects.

The tool has been implemented in Java, and follows the Extract-Query-View approach adopted by many reverse engineering tools [15]. In this approach we first extract the relevant data from the provided documents. This data, the work products and if available the reference traceability matrices, are stored in a database. For reconstructing the traceability links, queries can be done on the database. The reconstructed information combined with the data from the database is used to generate the requirements views.

6.1 Tool Requirements

In order to make ReqAnalyst useful in practice, it needs to fullfil the following requirements. One of the key requirements for ReqAnalyst is that it should reduce the effort for maintaining consistent traceability support and reduce the search time for changes, improve impact analysis and coverage analysis. Besides that, ReqAnalyst should be able to easily support different development environments and different domains with a minimum of tailoring effort. This includes environments such as global distributed software development, offshoring and outsourcing. Also the deployment of ReqAnalyst should be simple for such heterogeneous environments.

The input for ReqAnalyst consists of the work products that need to be traced and of which the requirements views should be generated. The tool should be flexible in the structure of these work products, minimizing the amount of tailoring required to offer a document as input to ReqAnalyst. In addition to that, it should be able to cluster the work products in an easy and flexible way.

Futhermore, ReqAnalyst should be scalable. It should be able to handle a large number of work products, but it should also be easily expandable with respect to the number of predefined requirements views (or other views, if necessary).

Since we anticipate that the maintenance of such a traceability matrix cannot be fully automated, ReqAnalyst should support manual traceability identification as well. In particular, it should be possible to read in a hand-written matrix, to compare the manual with the automatically obtained results, and to easily inspect the documents for which the two matrices differ.

In order to support the evaluation of reconstruction approaches, the latter comparison feature can be used for conducting a qualitative analysis of the reconstruction results. In order to support a quantitative analysis as well, the tools should be able to compute precision and recall figures from the traceability matrices.

6.2 Technology Used

ReqAnalyst is implemented using standard web-technology. For storing the data we use a MySQL[‡] database. On top of the database we have implemented a Java web application using Java Servlets (for

[‡] `http://www.mysql.com`

Case Studies	Pacman 2.2	Calisto	Philips
Number of Requirements (work products)	14	12	7
Number of Design Artifacts	24	48	16
Number of Test Cases	20	79	326
Total number of indexed terms	1366	2251	2502
Number of “requirement - design artifact” links	28	59	nk
Number of “requirement - test case” links	19	80	nk

Table 1: Case Study Statistics

collecting data and link reconstruction) and Java Server Pages (for presenting the results). The choice for building a dynamic web application in Java made it easy to fulfil a number of the practical tool requirements mentioned in the previous subsection, such as ease of deployment.[§] Furthermore, the use of a browser provides access to the tool from any location, making it suitable for global distributed software development.

6.3 Functionality and Implementation

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

A ReqAnalyst session starts by choosing a project, which can be a new one, or one that has been stored in the database already. Once the user has chosen a project, ReqAnalyst shows a menu with the steps that can be executed next.

ReqAnalyst first of all offers a menu to extract the data from the provided documentation. The work products and the reference traceability matrices can be extracted. Secondly, it provides a menu for setting the parameters of the LSI reconstruction and the choice for a link selection strategy.

Once the tool has executed a reconstruction an intermediate menu appears showing the reconstructed traceability matrix and some options for generating various requirements views. These views should make it possible to obtain continuous feedback on the progress of ongoing software development or maintenance projects. Futhermore, they facilitate communication between project stakeholders and different document owners. In addition to that, ReqAnalyst offers views that support the comparison of traceability matrices obtained in different ways, for example manual versus automatically via LSI. Examples are shown in Figures 7 and 8 discussed in the next section.

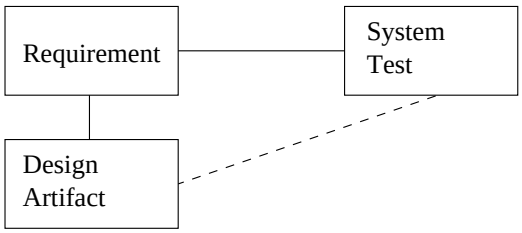


Figure 5: Traceability Model for our Case Studies

7 Case Studies

We have conducted three case studies where we applied our approach for reconstructing requirements traceability using LSI. The case studies vary in size and context. The first case study, Pacman, is a small case we developed within our university. This case study gives us the opportunity to explore all the possibilities of the techniques in a controlled environment. We varied the different parameters of our analysis to come to a setting giving the best results. The second case study, called Calisto, is somewhat bigger. Although developed within a university, the system at hand was developed for an external (industrial) client. The last case study involves an industrial project carried out at Philips Applied Technologies. This case study represents a real life project for commercial purposes.

In our case studies, we will focus mainly on two types of traceability links; links between requirements and design, and links between requirements and test. The corresponding traceability model is shown in Figure 5. By combining these two link types we can furthermore obtain traceability from design decisions to system tests, as indicated by the dashed line in the figure.

An impression of the size of the cases is provided by Table 1. It shows the number of work products involved relevant to our traceability model for each case, as well as the number of indexed terms for the total set of documents, including additional context (e.g. Javadoc). Besides that, it shows the number of links between the different work products as set in the provided reference traceability matrices[¶].

For each case study, we will conduct link reconstruction using the following link selection strategies: constant threshold, variable threshold, one dimensional vector filter and two dimensional vector filter, and reflect on the lessons learned from this case.

7.1 Case Study I: Pacman 2.2

Our first results are obtained from a lab experiment executed at Delft University of Technology. The system at hand is a simple version of the well-known Pacman game that is used by students in a lab course for testing object oriented software following Binder's testing approach [6]. An initial

[§]For our case studies we used the Apache Tomcat 5.5 web server for deployment

[¶]For the Philips case study, we do not have the reference tracebility matrices. So we do not know the number of links and cannot calculate the link density (nk – not known).

UC7	Suspend
Actor	player
1.	Entry condition: The player is alive and playing
2.	The player presses the quit button in order to suspend playing the game
3.	During suspension, no moves are possible, neither from the player nor from the monsters
3a.	The user can press undo in order to undo monster and player moves
4.	Pressing the start button re-activates the game

Figure 6: Full text for use case UC7 of the Pacman case study

implementation for the system is given, and students are expected to extend the test suite according to Binder's test design patterns, and enhance the system with additional features (which they should test as well).

7.1.1 Case Configuration

The available documentation for Pacman consists of

- A requirements specification including a concise domain analysis, ten use cases, and a description of the user interface.
- A design document listing the design decisions at architectural as well as detailed design level. This covers the (model-view-controller) layering used as reflected in the package structure, the static view explaining the main classes and their assocations, a dynamic view summarizing the system's main state machine, and a description of the implementation of the user interface.
- A testing document explaining the acceptance test suite for the application. For each use case one or more test cases are provided as well as a test case for validating the proper working of the user interface.

Pacman is shipped with a traceability matrix. As can be seen from the above description, Pacman's documentation has been organized with traceability in mind. Thus, for the acceptance test suite, there is a natural mapping from test case to use case. For the design it is somewhat harder to setup the documentation with clear traceability objectives. As an example, to what requirements should the decision to opt for a model-view-controller architecture be linked?

For the requirements specification the use cases are chosen as main requirement entities. Besides the use cases we also included the domain analysis, user interface, and the requirements for the development environment. The design is listed according to its design decisions, which we used as design entities in our analysis. Finally, every test case is considered as a test case entity for our analysis. In total there are 14 requirement entities, 24 design artifacts, and 20 test cases. In Figure 6 we show an example of an use case description. The documents were provided in plain text, and the traceability matrix as an MS Excel spreadsheet. They could be directly passed as input to ReqAnalyst.

As corpus, the collection of all documents was used, including the Javadoc of the implementation. This resulted in a corpus of almost 1366 terms. Furthermore, for c we took the value 0.7. The other two values k and q we varied to get an impression of the impact of these values.

7.1.2 Case Results

The recall (R) and precision (P) for this case study are shown in Table 2 for the constant threshold, variable threshold, one dimensional vector filter and two dimensional vector filter strategies discussed in Section 5. For the two dimensional vector filter strategy we also recorded the link density (LD). In Figure 7 and Figure 8 the reconstructed traceability matrices of the Pacman case study are shown using the various filter strategies. Figure 7 shows the reconstructed matrices of the links between the requirements and the design, and Figure 8 shows the reconstructed matrices of the links between the requirements and the test cases.

The reconstructed matrices are compared with the provided reference traceability matrix. The correctly reconstructed links are colored grey and each the cell contains an "X". The empty cells are correctly *not* reconstructed links. According to the reference traceability matrix these cells should not return a link.

The cells that are colored light grey are invalid compared to the provided reference traceability matrices. These cells containing "fp" are the false positives. These links should not be reconstructed as traceability links and are therefore incorrectly reconstructed. The dark grey cells containing "fn" are the false negatives (missing links). A link should have been reconstructed between these two particular work products, but our approach did not select this candidate link as a traceability link. In ReqAnalyst, each cell in these matrices is clickable leading to the text of both documents. This makes it easy to analyze why a certain reconstructed link was present or absent.

7.1.3 Results "Requirements – Design"

The results in Table 2 show a relatively low precision of the links between the requirements and design. This is caused by the many false positives. The constant threshold strategy returns the most false positives (and this way returns the lowest precision). The threshold of $c = 0.7$ has almost no influence on the result (see Figure 7a). Most similarity measures are above 0.7.

If we apply the variable threshold strategy we filter many of the false positives. This strategy generally increases the precision, but decreases the recall, e.g., for $q = 30\%$. Figure 7b shows that 5 correct links are filtered using these settings. We can also see that many of the false positives are located in specific rows and columns. In the case of $q = 30\%$, design artifacts D0, D3.3 and D3.7 and requirement artifacts DA, UC7 and GUI return many false positive.

Our one dimensional vector filter strategy filters many of these false positives in the columns of the traceability matrix. For example for the column with label DA (Domain Analysis) it filters an additional 8 false positives compared to the variable threshold strategy (see Figure 7b and Figure 7c). The same can be observed for UC7 (4 additional false positives) and GUI (6 additional false positives). In this case, with $q = 30\%$ the filter increases the precision and does not influence the recall (see Table 2 with $q = 30\%$). However, the filter has limited influence on the rows containing many false positives such as D0, D3.3 and D3.7.

			Constant Threshold		Variable Threshold		One Dimensional		Two Dimensional	
Link type	*c*	*q*	R	P	R	P	R	P	R	P
Use case to design	0.7	10%	1.0	0.09	0.36	0.25	0.29	0.13	0.21	0.15
	0.7	20%	1.0	0.09	0.54	0.15	0.64	0.17	0.46	0.17
	0.7	30%	1.0	0.09	0.82	0.13	0.82	0.16	0.71	0.17
	0.7	40%	1.0	0.09	0.93	0.11	0.89	0.13	0.82	0.14
	0.7	50%	1.0	0.09	0.93	0.09	0.93	0.11	0.86	0.12
	0.7	60%	1.0	0.09	1.0	0.09	0.97	0.10	0.93	0.10
Use case to test	0.7	10%	1.0	0.07	0.42	0.36	0.58	0.27	0.53	0.42
	0.7	20%	1.0	0.07	0.74	0.24	0.68	0.19	0.68	0.27
	0.7	30%	1.0	0.07	0.95	0.17	0.84	0.18	0.79	0.21
	0.7	40%	1.0	0.07	0.95	0.13	0.95	0.14	0.95	0.16
	0.7	50%	1.0	0.07	0.95	0.10	0.95	0.11	0.95	0.13
	0.7	60%	1.0	0.07	1.0	0.09	0.95	0.10	0.95	0.11

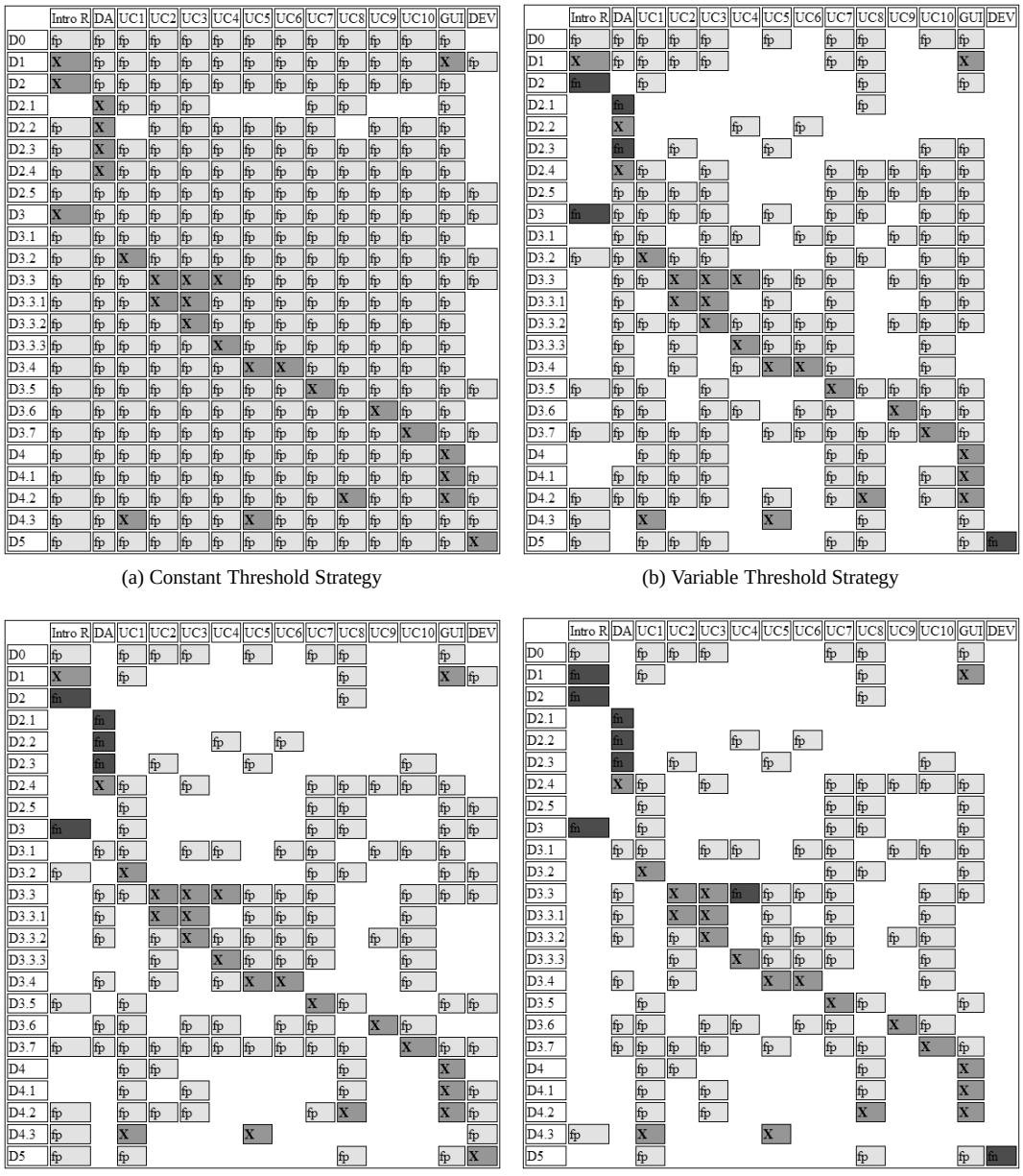
Table 2: Recall and precision for the reconstructed traceability matrices of Pacman 2.2 with rank-*k* subspace of 20% and $c = 0.7$

Using our two dimensional vector filter strategy also affects the rows of the matrix. Compared with the one dimensional vector filter strategy we filter an additional 1 false positive for D0, 3 false positives for D3.3, and 4 false positives for D3.7. In this case we did increase the precision a little, but also decreased the recall; 3 correct links are now filtered (dark grey cells containing "fn").

The two dimensional vector filter strategy did also filter one additional false positive in UC7. Still UC7 contains many false positives. The quantitative analysis did not help us to understand this phenomenon so we needed to explore the text. We used the "browse results view" of ReqAnalyst for this. We investigated the text of the correct links and the returned links with the best score. We manipulated the text to improve our understanding of these links. Improving the similarity measure of the correct links was not that difficult, but understanding why the other links had such a high similarity score was not always that obvious.

To improve the correct similarity measure of UC7 (See Figure 6) the state conditions were made more explicit in the design text. So documenting that a state has changed, e.g., to "playing state" again, is not sufficient. Explicitly documenting that the player is "alive and playing" helps to link the design artifact to the use case.

Furthermore, in the design artifact the term "pause" was used for indicating a suspension. So we also introduce the term "pause" in the use case description. The last step of the use case description was changed in: "4. Pressing the start button ends the pause, and re-activates the game". These changes in the text increased the similarity measure of the correct link. However, this did not influence the total result of use case UC7. UC7 still returned 12 false positives. The other similarity measures did not sufficiently decrease for the link selection strategy to filtered them.



(a) Constant Threshold Strategy

(b) Variable Threshold Strategy

(c) One Dimensional Vector Filter Strategy

(d) Two Dimensional Vector Filter Strategy

Figure 7: Reconstructed traceability matrices between requirements and design using different link selection strategies with rank-k subspace of 20%, $c = 0.7$ and $q = 30\%$

7.1.4 Results "Requirements – Test"

Looking at the links between the requirements and test cases we observed similar results. The constant threshold strategy does not have much influence and only filters a small number of candidate links resulting in many false positives (see Figure 8a).

The variable threshold strategy does a much better job and filters many of the false positives. Again we can see that a number of rows and columns cause the many false positives. In this case the rows Intro TC, TC6 and TC11, and the columns DA, UC10 and GUI (see Figure 8b).

Our one dimensional vector filter strategy again filters many additional false positives, but in this case it also filters some correct links causing the recall to decrease (see Table 2 with q = 30%). Two correct links are filtered (see Figure 8b and Figure 8c). For the variable threshold strategy the threshold ε = 0.88. For the column UC5 the threshold ε = 0.91, and for column UC10 the threshold ε = 0.92. So, for both UC5 and UC10 the threshold is higher filtering more cells in that column. Cell $\langle UC5, TC5a \rangle$ has a similarity of 0.9 (< 0.91) and because of that it will be (incorrectly) filtered using the one dimensional vector filter strategy. The same holds for cell $\langle UC10, TC10a \rangle$, which has a similarity of 0.91 and the threshold for that column is 0.92.

The two dimensional vector filter strategy shows the expected result. It filters some additional false positives in the rows of the matrix increasing the precision. Unfortunately, again one additional correct link is filtered (see Figure 8c and Figure 8d).

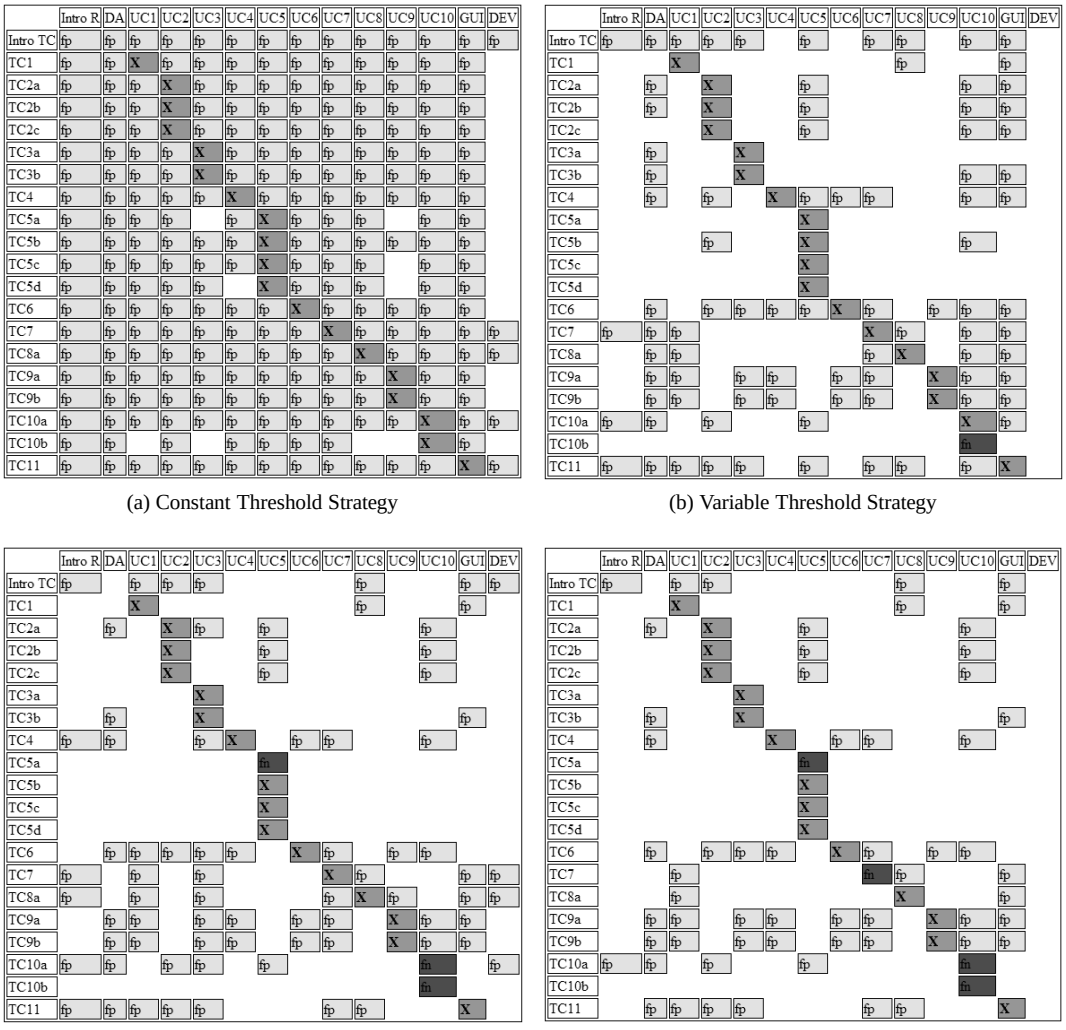
7.1.5 Lessons Learned

The key lessons learned from this case study are:

- Reconstructing traceability between use cases and test cases is easier than between use cases and design.
- The design activity and traceability activity is a hard combination. The designer should structure the design decisions so that clear traceability can be established.
- For larger case studies we do not expect results to become better than for Pacman. Pacman is designed to incorporate traceability and for most industrial projects this only limitedly done [19, 20].
- Eliminating false positives in columns with many hits is effectively done by the one dimensional vector filter strategy.
- Eliminating false positives in columns, as well as rows with many hits is effectively done by the two dimensional vector filter strategy.

7.2 Case Study II: Calisto

In this section we discuss our results from the second case study. This case study involves software developed by students from Eindhoven University of Technology in a software engineering project where the students needed to carry out a complete development life-cycle.



(a) Constant Threshold Strategy

(b) Variable Threshold Strategy

(c) One Dimensional Vector Filter Strategy

(d) Two Dimensional Vector Filter Strategy

Figure 8: Reconstructed traceability matrices between requirements and test cases using different link selection strategies with rank-k subspace of 20%, $c = 0.7$ and $q = 30\%$

In this project an Interface Specification Tool is constructed. This tool is designed to support the ISpec approach, a specification method in the context of component technology [27]. The purpose of the tool is to create Interface Specification Documents as well as exporting these documents to other software components.

7.2.1 Case Configuration

The provided documentation for Calisto consists of:

- A user requirements specification (URD), which states what the product is supposed to do according to the client. It is used as the contract between the client and the developers of the product.
- A software requirements specification (SRS), which formally describes the functionality of the product to be made. The document translates all user requirements into software requirements. It defines a logical model that contains the functionality that was found in the user requirements. The functional requirements are described using the classes defined in the logical model including attribute and method descriptions.
- An acceptance test plan (ATP), which describes the plan for testing the developed software tool against the user requirements. It lists the test cases that should cover the user requirements.

The documents all comply with the equally named specifications from the Software Engineering Standard, as set by the European Space Agency (ESA) [1]. We consider the SRS as a design document as it specifies classes and interfaces. Thus, in our analysis we will refer to the software requirements as our design artifacts.

All requirements have a unique identifier; the user requirements comply to the prefix URCARxx and the software requirements to the prefix SRFURxx were xx is a unique number. The test cases are directly related to the user requirements as they have the same unique identifier, namely URCARxx.

We did the analysis including with the code included in as well as excluded from the corpus. In the first case the corpus consisted of almost 5500 terms, in the second case it consisted of almost 2300 terms. The second case did contain additional context from the provided documents. This additional text includes the introductions to specific groups of requirements or “glue” text to make the document readable and not just a list of requirements. In this paper we discuss the results of the second case not including the code. We started with the same values for k, c and q as in the Pacman case.

7.2.2 Case Results

The precision and recall for all link selection strategies for the Calisto case study are summarized in Table 3. In this case study we observed that the constant threshold has a major impact on the results. When using the commonly accepted threshold of $c = 0.7$ LSI returns only few links. Using a threshold of $c = 0.4$ makes that the constant threshold has almost no influence on the results, but gives the best results. Filtering only on the constant threshold ($c = 0.4$) will cause the recall of design never to exceed 0.54 and the recall of test never to exceed 0.94.

Remarkable is the difference between the variable threshold strategy and one dimensional vector filter strategy for both link types. This can be explained by the distribution and the stretch in the

			Constant Threshold		Variable Threshold		One Dimensional		Two Dimensional	
Link type	*c*	*q*	R	P	R	P	R	P	R	P
Requirements to design	0.4	10%	0.54	0.12	0.07	1.0	0.19	0.39	0.15	0.69
	0.4	20%	0.54	0.12	0.10	0.60	0.27	0.28	0.22	0.43
	0.4	30%	0.54	0.12	0.15	0.35	0.41	0.21	0.31	0.29
	0.4	40%	0.54	0.12	0.31	0.26	0.51	0.17	0.44	0.27
	0.4	50%	0.54	0.12	0.41	0.15	0.53	0.14	0.49	0.18
	0.4	60%	0.54	0.12	0.58	0.12	0.54	0.13	0.51	0.15
	0.3	50%	0.69	0.11	0.41	0.15	0.63	0.14	0.54	0.19
	0.3	60%	0.69	0.11	0.58	0.12	0.68	0.12	0.58	0.15
Requirements to test	0.4	10%	0.94	0.16	0.05	1.0	0.23	0.44	0.20	0.70
	0.4	20%	0.94	0.16	0.16	1.0	0.51	0.41	0.45	0.69
	0.4	30%	0.94	0.16	0.36	0.69	0.71	0.28	0.61	0.50
	0.4	40%	0.94	0.16	0.60	0.35	0.83	0.21	0.75	0.37
	0.4	50%	0.94	0.16	0.79	0.20	0.85	0.17	0.79	0.25
	0.4	60%	0.94	0.16	0.96	0.14	0.94	0.16	0.89	0.20

Table 3: Recall and precision for the reconstructed traceability matrices of Calisto with rank-*k* subspace of 20%

data set. For example when we take the reconstructed links between requirements and design for *q* = 10%. In the case of appyling the variable theshold strategy the threshold $\varepsilon = 0.86$ explaining the low recal and high precision. When applying the one dimensional vector filter strategy the $\varepsilon = 0.57$ for a specific column. The lower threshold return more links increasing the recall and decreasing the precision compared to the variable threshold.

As for Pacman, we can see that the precision obtained using our two dimensional vector filter strategy is higher in all cases – in fact the improvement is even higher than we had for the Pacman case. However, the recall was consequently lower with respect to the first strategy using similar parameters. This can be explained as follows. First, again we have certain design artifacts containing many false positives. For example, one has 7 and another has 5 false positives. The second strategy reduced the number of false positives to 0 for the first case (causing the increase in precision). For the second case 4 false positives are filtered, but in this case also 2 correct links are filtered. This causes the recall to decrease.

When looking at the results of the reconstruction of the links between the requirements and design we can identify a separation between the part that describes the classes (functionality) and the part that describes the additional non-functional aspects such as portability and maintainability. In the part that describes the functionality of the system we have quite a few missing links (causing the low recall). In the part that describes the non-functional aspects we have many false positives (causing a decrease in precision). Looking at the text we see that the structure of the description is not that different, so this cannot be the reason for this separation. The cause for this separation should then be in the description

itself. Indeed the non-functional aspects are more extensively described by text, as the classes are more described by diagrams, pseudo-code and data types.

7.2.3 Lessons Learned

- Reconstructing traceability between requirements and test cases again performs better than between requirements and design.
- In this case the description of the design was often captured in diagrams, pseudo-code or datatypes. This information is ignored by our analysis emphasizing the difficulties of treacing requirements to design.
- Eliminating columns with many hits is effectively done by the one dimensional vector filter strategy.
- Eliminating rows with many hits is effectively done by the two dimensional vector filter strategy.
- The Software Engineering Standard by the European Space Agency influences the choice for the work products to be analysed and has a direct impact on the result (see descripsion of functional and non-functional aspecs)
- It is indeed hard to get better results in a real-life industrial project compared to the Pacman case study. However, the results for "requirements – test" return comparible results in both case studies. With a similar recall the precision for Calisto is even better.

7.3 Case Study III: Philips Applied Technologies

For most products Philips Applied Technologies develops, almost 80–90% is reused from previous projects. The majority of new products has only limited new functionality that needs to be developed from scratch. The existing functionality is delivered by various Philips units.

In this case study the document set of an extension of a DVD+RW recorder is analyzed for requirements coverage. We want to know if all the requirements agreed in the contract are covered in the product. That is, we trace the requirements in the rest of the work products.

During product development a large number of requirements initially identified cannot be traced back to test cases or design documents: in a way they "get lost". This gets even worse when the system evolves over time. First ad-hoc attempts in two case studies showed that less than 10% of the total requirements can be recovered from the design and test documents (see Section 7.3.2). Furthermore, as the system evolves, new requirements are introduced in the system that cannot be traced back to the original requirements specifications.

7.3.1 Case Configuration

In this case the total set of documentation consists of one general document, which describes the document structure for this component. Furthermore there is one requirements document, which describes the requirements of the component, and an architecture document, which describes the delta that is introduced due to the new functionality. Finally, there are 5 interface specifications, 11

component specifications, which together form the design of the component and one test specification containing all the test scenarios and test cases.

In total, 20 documents are analyzed in this case study. These documents are all Microsoft Word documents and are based on the IEEE-1233-1998 standard for system requirements specifications [25]. Furthermore, they are not explicitly related. Thus, no reference traceability matrix is provided or anything comparable.

In this case it was not very obvious how to identify the concepts in the documentation. The requirements all have a unique identifier, but one of the problems is that the requirements specification consists of a requirements hierarchy. We need to choose the right granularity for our requirement concept. In this case study we took the highest level of requirements including their sub-levels as documents in the LSI analysis. This resulted in 7 high-level requirements as input for the LSI analysis.

For design artifacts our choice was not very obvious either. Every document contains the design of one component or interface in the system. Taking a component as design artifact makes sense as the internal structure of the design documents is not really suitable for subdividing into smaller parts. So each design document will be a design artifact for the LSI analysis. In total this makes it 16 design artifacts.

The identification of concepts in the test specification was not really difficult. Test scenarios and test cases were easily recognizable in the test specification and therefore very suitable as input for the LSI analysis. In total we have 326 test cases. After preprocessing the documents this resulted in a corpus of more than 2500 terms representing the engineering domain.

7.3.2 Preliminary Analysis of Documents

Before we executed the LSI analysis on the document set we carried out a simple exploring analysis on the documents using Xpath expressions [8]. In practice often simple search facilities are used to reconstruct links, for example, when a new requirement is agreed and needs to be processed in the documentation [43]. In this first analysis we transformed the Microsoft Word documents to XML format and did some searching on the document set using Xpath expressions. Somewhat surprisingly, this analysis showed no direct links between the requirements documents and any other document. The unique identifiers were not traceable in the rest of the documents. Querying with the labels of a requirement identified only few links.

Still traceability should be incorporated somehow in the document set; after all this is the documentation set of one and the same product extension. Taking a closer look at the documents showed that this is indeed true. When analyzing the documents and focusing on a specific requirement it showed that this requirement is transformed to a different technical term during architectural design. Components in the architecture get a name not directly related to the requirement.

From this experiment we learned that it is often very hard for non-experts to understand the documentation and use it effectively. This preliminary analysis emphasizes again that you need an expert to set up the traceability meta-model (define the concepts that need to be traced) and to judge the reconstructed traceability matrix.

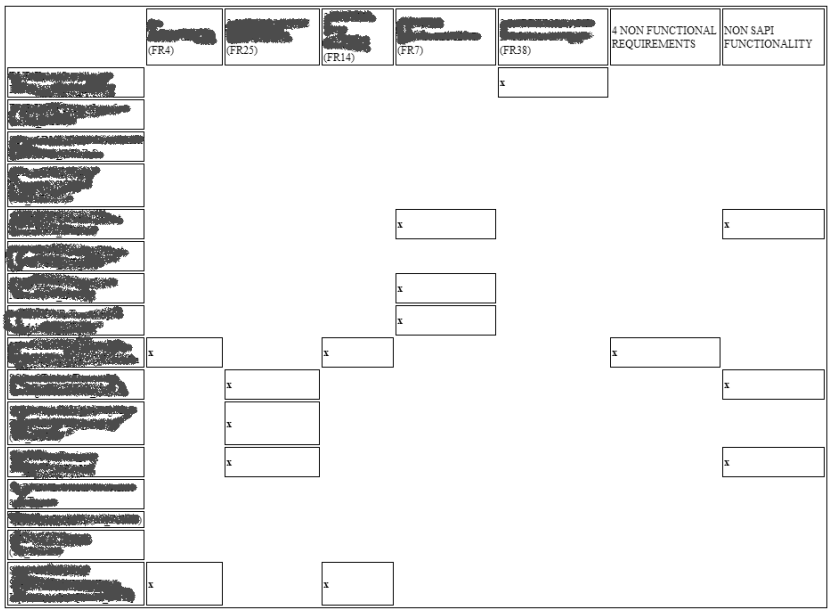


Figure 9: Reconstructed Traceability Matrix for Philips with a rank-*k* subspace of 20% and applying the Two Dimensional Vector Filter Strategy using $c = 0.4$ and $q = 20\%$

7.3.3 Case Results

Executing the LSI analysis resulted in more informative results. The first remarkable result is that the similarity measures between the requirements and the design were much better than the similarity measures between the requirements and the test cases. The first two case studies showed the opposite results. A reason for this is the choice of the granularity of the concepts for analysis; high-level requirements and complete design components in combination with the structure of these documents. Every design component starts with a general description of the component. Part of that general description includes its functionality. This functionality matches the functionality described in the requirements description. Figure 9 shows a reconstructed traceability matrix for the requirements and design components. It has a link density of 13% and should give a direction for the experts to find their requirements.

The similarity measures between requirements and test cases showed results that were not as good. In this case there is a mismatch in the granularity of the documents used for analysis. The requirements were the same high-level requirements, but the test cases can be considered more as unit tests. These unit tests are written according to the designs and not the requirements. Clustering the 325 test to 16 higher level "test categories", did not improve the reconstruction results. The quality of the similarities between de requirements and test cases was insufficient. It seems that the terminology has changed during the design phase making it difficult to reconstruct traceability between requirements and test.

Finally, in this case we were not able to compare the results with a provided traceability matrix, so we had to consult experts knowing the system. Disappointing is the fact that it is hard to validate that the reconstructed links are indeed correct. We found several links that are correct, but we did not come to an agreement for all links. For this reason we could not calculate the recall and precision.

7.4 Lessons Learned

The key lessons learned from this case study are:

- The choice for the work products to be traced is essential. The expert needs to decide on the work products that are most suitable to trace and the level of granularity to get the best results.
- The IEEE-1233-1998 standard for system requirements specifications shows a possibility to improve the reconstruction of traceability links between requirements and design. In IEEE-1233-1998 it is mandatory to provide a general description of the component.
- Again we see that is it hard to get better results in a real-life industrial project compared to the Pacman case study, which perhaps can be considered as an upper bound of the quality that LSI-based link reconstruction can achieve.

8 Discussion

8.1 Link Selection

With respect to the link selection strategies it is very hard to conclude when a strategy performs better. It very much depends on the application objectives. In the case studies we showed the results of applying

LSI using different parameter values and compared the available link selection strategies. If we set our objective to realizing 100% recall, we can say that our two dimensional vector filter strategy performs best. With a recall of 100% the corresponding precision is higher for all cases.

We have also seen that the strategies cannot always reduce the number of false positives successfully. A good example of this is use case UC7 in the Pacman case. The returned links were all quite similar (between 0.91 and 0.96). The infinity of possible links in our strategies can be a disadvantage. In this case the cut point and cut percentage strategy will be more successful. They will, regardless of the actual value, simply select only the best x similarity measures (where x is a natural number).

Another point of attention is the constant threshold, which can also be disadvantageous. The constant threshold protects against losing too much quality in similarity measures, since every link should at least have a similarity measure $\geq c$. In some cases a correct link will be filtered only because of the constant threshold. Changing the values for the variable threshold will not help to recover these links. In this case a recall of 100% can never be reached as can be seen in the Calisto case and in the Philips Applied Technologies case concerning the requirements coverage in the test cases. In this case all similarity measures are simply lower than the constant threshold. The opposite is also possible, in the Pacman case the constant threshold was hardly of any influence.

A solution can be to make the constant threshold dependent on the minimal and maximal similarity measures or the mean of the entire matrix. This way it is more related to the actual output and not chosen randomly. It also represents the quality of the data. If for example the maximum similarity measure of the matrix is 0.67 and the constant threshold is 0.7 no links will be found with both strategies. Taking the mean of the total data set as constant threshold ensures links will be found. Note that this does not mean that for every requirement a link will be found, so for individual requirements the idea behind the constant threshold is kept. The analist together with the expert should decide on the quality of the data set.

8.2 Link Types

Futhermore we observe a systematic difference in the performance of LSI for the different link types; requirements in design and requirements in test cases. The links between requirements and test case in general performed better than the links between requirements and design. The Philips Applied Technologies case was an exception, which can be explained by the wrong choice of granularity of the test cases. In general, the reason why test cases perform better is unclear and is still an open issue that remains to be answered.

However, one of the reasons LSI performs worse for “requirement – design” relations is the fact that for designs many diagrams are used for documentation (see again [51]). Additionally, the diagrams are often badly described. Information retrieval techniques are less suitable for traceability reconstruction in this case. So, it very much depends on the provided document structure. If many diagrams are used to capture the information of the design, these should also be accompanied with a clear description. When defining the traceability model these things need to be considered and decided upon.

8.3 Link Density

We expect that there is a general range in the number of links that should be in an “ideal high quality” traceability matrix. We call this metric the link density for a traceability matrix of size $N \times M$.

Case Studies	Pacman 2.2	Calisto	Philips
Number of Requirements (work products)	14	12	7
Number of Design Artifacts	24	48	16
Number of Test Cases	20	79	326
Number of "requirement - design artifact" links	28	59	nk
Number of "requirement - test case" links	19	80	nk
Link density between "requirement - design artifact" links	0.08	0.10	nk
Link density between "requirement - test case" links	0.07	0.08	nk
Link density relation "requirement:design artifact"	1:1,7	1:4	nk
Link density relation "requirement:test case"	1:1,4	1:6,6	nk

Table 4: Link Density Statistics

Initially, we calculated the link density by dividing the total number of links set in the reference traceability matrix by the total number of links that can be set (the total set of possible candidate links). For example, in our Pacman case we have 28 "requirement - design artifact" links and we divide that number by 336 (14 requirements x 24 design artifacts). For both the Pacman and the Calisto case study we calculated the link density [‖]. We see that the link density for our case studies is always between 7% and 10% (see Table 4). This observation can be an indication that the link density should always be situated between the 5% and 15% of the total candidate links when doing "adequate" traceability. Maybe this link density can be used as a guideline for traceability reconstruction. For example, in the Philips case where we do not have a reference traceability matrix. If you reconstruct a traceability matrix and the link density is 30%, it means that around 15–20% of the returned links are probably false positives.

This initial number gives only a general view on a traceability matrix. It gives an indication of how many links there should be in total. This initial number does not give any direction on the structure of the traceability matrix, while in most cases we know that a traceability matrix is structured around its diagonal; the first requirement is implemented in the first design artifact and tested by the first test case, and so on...

In the ideal case we want to know the specific relation of the link density between the requirements and all other work products (design artifacts and test cases). For example, each requirement has on average a link with 2 test cases. If we assume this number is constant, we know that every additional requirements has 2 corresponding test cases. In Table 4 we calculated this link density relation for our case studies. It shows that each requirement in the Calisto case study has on average 4 links to a design artifact. Future work should confirm if there is indeed an "ideal" constant link relation between requirements and other work products as we propose here.

[‖] Again for the Philips case study we cannot calculate these numbers as we do not have a reference traceability matrix (nk – not known).

8.4 Reserach Issues

On the basis of our case studies we can identify the following research issues that need to be addressed before LSI can be applied for the purpose of link reconstruction in an industrial context:

- How can we take advantage of the (few) cross references that are typically included already in the documents? For example, does it make sense to give requirement identifiers used in design documents a high weight in the term-by-document matrix?
- How should the hierarchical structure of, for example, design or requirements documents be dealt with in link reconstruction? In our present experiments we created a flat set of documents. How can LSI be adjusted so that if possible links in the most detailed documents are taken into account, moving up higher in the hierarchy (aggregating the paragraphs) when this does not lead to a sufficient number of results?
- What documents should be included in the corpus? For example, do we get better requirements-to-design links when we also include the test documents in the corpus? Why (not)?
- Can we come up with link reconstruction techniques tailored towards specific types of links? For example, do we need different strategies for reconstructing requirements-to-design and for requiements-to-test links?
- Are the recall and precision that are achieved sufficient for practical purposes? Can we make sufficiently accurate predictions of certain coverage views based on the (incomplete) links we can reconstruct?
- Is the link density a good measure to characterize a “good” traceability matrix? If so, what will be the range for the number (5–15%) for various link types?

9 Concluding Remarks

The objective of the paper is to investigate the role latent semantic indexing can play in order to reconstruct traceability links. From the previous discussion we can conclude that LSI can indeed help increasing the insight in a system by means of reconstructing the traceability links between the different work products produced during development. We consider following to be our main contributions:

- We have provided a methodology, MAREV, for automating the process of reconstructing traceability and generating requirements views.
- We defined a new two-dimensional vector filter strategy for selecting traceability links from an LSI similarity matrix.
- We provided a tool suite, ReqAnalyst, for reconstructing traceability links including support for quantitative and qualitative assessment of the results.
- We applied our approach in three case studies of which one was an industrial strength case in the consumer electronics domain.

- For each of the case studies, we offered an analysis of factors contributing to success and failure of reconstructing traceability links.
- We identified the most important open research issues pertaining to the adoption of LSI for link reconstruction purposes in industry.

Our future work will be concerned with the open issues listed in the discussion section. Furthermore, we would like to extend our work along three lines. First, we want to study other links than those between requirements on the one hand and test cases and design decisions on the other. Furthermore, we are in the process of extending our experimental basis. In particular, we are working on two new case studies in the area of consumer electronics and traffic monitoring systems. In these case studies we want to focus more on our last step: the generation of requirements views that are useful in practice. Finally, we will further explore how to improve the performance of LSI and the link selection strategies in real-life applications. All can be implemented in ReqAnalyst.

Acknowledgments We would like to thank Lou Somers of Eindhoven University of Technology for his cooperation and providing us the Calisto project work products.

Furthermore we would like to thank the ITEA organization for enabling and supporting the MERLIN collaboration. In particular, we would like to thank Rob Kommeren of Philips Applied Technologies for his cooperation and providing us an industrial case study.

Partial support was obtained from NWO Jacquard, project Reconstructor, and SenterNovem, project Single Page Computer Interaction (SPCI).

References

[1] European Space Agency. ESA software engineering standards (ESA PSS-05-0 Issue 2). Technical report, ESA Board for Software Standardization and Control (BSSC), 1991.

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

[3] 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.

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

[5] M. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[6] R. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley, 2000.

[7] Sjaak Brinkkemper. Requirements engineering research the industry is and is not waiting for. In *Proc of the 10th Int. Workshop on Requirements engineering: Foundation for Software Quality*, 2004.

[8] James Clark and Steve DeRose. XML path language (XPath) version 1.0. Technical report, W3C, November 1999.

[9] Jane Cleland-Huang, Carl K. Chang, and Jefrey C. Wise. Automating performance-related impact analysis through event based traceability. *Requirements Engineering*, 8(3):171–182, August 2003.

[10] 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.

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

[12] 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.

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

[14] 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, 2004.

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

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

[17] 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.

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

[19] O. Gotel and A. 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.

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

[21] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Proc. of the 11th IEEE Int. Conf. on Requirements Engineering*, page 138, Washington, DC, USA, 2003. IEEE Computer Society.

[22] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Improving after-the-fact tracing and mapping: Supporting software quality predictions. *IEEE Software*, 22(6):30–37, 2005.

[23] 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.

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

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

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

[27] H. B. M. Jonkers. Ispec: Towards practical and sound interface specifications. In *Proc. of the 2nd Int. Conf. on Integrated Formal Methods*, pages 116–135, London, UK, 2000. Springer-Verlag.

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

[29] Patricia Lago and Hans van Vliet. Explicit assumptions enrich architectural models. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 206–214, 2005.

[30] 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.

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

[34] 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.

[35] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Adams re-trace: A traceability recovery tool. In *Proc. of the 9th European Conf. on Software Maintenance and Reengineering*, pages 32–41. IEEE Computer Society, March 2005.

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

[37] 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.

[38] A. Marcus, J.I. Maletic, and A. Sergeyev. Recovery of traceability links between software documentation and source code. *Int. Journal of Softw. Eng. and Know. Eng.*, 15(5):811–836, October 2005.

[39] 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.

[40] 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.

[41] 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.

[42] Johan Natt och Dag, Vincenzo Gervasi, Sjaak Brinkkemper, and Bjorn Regnell. Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In *Proc. of the 12th Int. Conf. on Requirements Engineering Conference*, pages 283–294, Washington, DC, USA, 2004. IEEE Computer Society.

[43] 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.

[44] 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.

[45] M. Di Penta, S. Gradara, and G. 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.

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

[47] 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.

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

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

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

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

[56] 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-2007-007
ISSN 1872-5392