Can LSI help Reconstructing Requirements Traceability in Design and Test?

Lormans, M.; van Deursen, A.

Publication date
2006

Document Version
Accepted author manuscript

Published in
Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR)

Citation (APA)
Lormans, M., & van Deursen, A. (2006). Can LSI help Reconstructing Requirements Traceability in Design and Test? In G. Visaggio, G. A. Lucca, & N. Gold (Eds.), *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR)* (pp. 47-56). IEEE.

Can LSI help Reconstructing Requirements Traceability in Design and Test?

Marco Lormans
Delft University of Technology
M.Lormans@ewi.tudelft.nl

Arie van Deursen
CWI and Delft University of Technology
Arie.van.Deursen@cwi.nl

Abstract

Managing traceability data is an important aspect of the software development process. In this paper we investigate to what extent latent semantic indexing (LSI), an information retrieval technique, can help recovering the information needed for automatically reconstructing traceability during the development process. We experimented with two different link selection strategies and applied 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.

1. Introduction

For many organisations the purpose of requirements management is to provide and maintain clear agreements between the different parties involved in developing the product, while still allowing requirements evolution. Managing requirements in such a way that useful information can be extracted from this requirements set, is very hard in practice [13]. This extracted information can be used for many applications such as generating requirements views [16] or impact analysis [8]. Examples of requirements views 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.

Up-to-date information retrieval 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 process demanding disciplined developers [6, 12, 17]. Currently available tool do not support the feature of automatically recovering traceability links [2].

In this paper we seek to investigate to what extent relevant traceability links can be reconstructed automatically from available documents using latent semantic indexing (LSI). LSI is a promising information retrieval method assuming there is a latent semantic structure for every document set [9]. For this reason, our approach assumes a document-oriented requirements engineering process, 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 much 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, aimed at answering the following questions:

1. Can LSI help in reconstructing meaningful requirements—design and requirements—test case traceability relations?
2. What is the most suitable strategy for mapping LSI document similarities to reconstruction links?
3. What are the most important open issues that need to be resolved before LSI can be applied successfully in an industrial context?

The three case studies we applied LSI to 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 Java unit tests. 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. 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 the link reconstruction method we used as well as the tools we developed to support our process. The three cases studies are presented in Section 5. In Section 6 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 and Traceability

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 [23]. This work is based on the viewpoints framework presented by Finkelstein *et al.* in [11]. Nissen *et al.* show that metamodels help managing these different requirements perspectives [22].

An important area of research in the area of traceability is developing these metamodels. These so called reference models discussed in [20, 26, 34] define the development artifacts including their attributes, and the traceability relations that are allowed to be set between these artifacts.

Von Knethen for example proposes (conceptual) traceability models for managing changes on embedded systems [32]. 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 [17]. 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 [10].

2.2. Link Reconstruction

To reconstruct coverage views from project documentation we need some traceability support. Several traceability recovery techniques already exist each covering different traceability issues. 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 use latent semantic indexing for recovering the traceability relations between source code and documentation [21]. The IR methods in these cases are mostly applied for 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 [14, 24]. In these cases traceability recovery is mainly used for managing the requirements after development when all the documentation needs to be finalized and released.

De Lucia *et al.* present an artifact management system, which has been extended with traceability recovery features [18, 19]. This system manages all different artefacts 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.

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 [25].

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 [9]. This is particularly caused by classical IR issues as *synonymy* and *polysemy*. LSI uses statistical techniques to estimate this latent structure. A description of terms and documents based on the underlying latent semantic structure is used for representing and retrieving information. 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 [4, 29]. Any rectangular matrix, for example a **t** x **d** matrix of terms and documents, **X**, can be decomposed into the product of three other matrices:

$$\mathbf{X} = \mathbf{T_0 S_0 D_0'}$$

such that $\mathbf{T_0}$ and $\mathbf{D_0}$ have orthonormal columns and $\mathbf{S_0}$ is diagonal. This is called the singular value decomposition of **X**. $\mathbf{T_0}$ and $\mathbf{D_0}$ are the matrices of left and right singular vectors and $\mathbf{S_0}$ is the diagonal matrix of singular values.

SVD allows a simple strategy for optimal approximate fit using smaller matrices. If the singular values in $\mathbf{S_0}$ 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 $\mathbf{X}'$ which is only approximately equal to **X**, and is of rank **k**. Since zeros were introduced into $\mathbf{S_0}$, the representation can be simplified by deleting the zero rows and columns of $\mathbf{S_0}$ to obtain a new diagonal matrix **S**, and deleting the corresponding columns of $\mathbf{T_0}$ and $\mathbf{D_0}$ to obtain **T** and **D** respectively. The result is a reduced model:

$$\mathbf{X}' = \mathbf{TSD}' \approx \mathbf{X}$$

which is the rank-k model with the best possible least square fit to **X** [9].

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 [9]. Our choice will be discussed when we apply LSI in our case study.

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 document 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 [18,20].

4. Approach

In order to answer the questions raised in the introduction, we conducted three case studies, which are described in Section 5. In the present section we discuss 1) the approach we used to reconstruct the traceability links 2) the approach we used to assess the reconstructed links, and 3) the tools that we developed in order to carry out these steps.

4.1. Link Reconstruction Steps

In [16] we have proposed an approach for reconstructing these requirements coverage views. In this particular case we experimented with LSI. This resulted in reasonably good traceability recovery results. The steps are [16]:

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 coverage views.

In this paper we will focus on step 4 and 5 handling the traceability recovery and selection of correct links. Of course, step 1 and 2 are of major importance for executing step 4 and 5 successfully. We will discuss these steps shortly and then focus on the other steps.

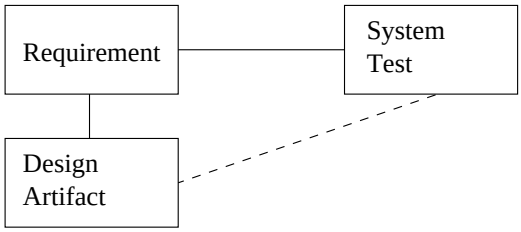

Figure 1. Traceability Model

4.1.1. Traceability Model and Concept Identification

In our case studies we will focus mainly on two types of traceability links; links between requirements and design, and requirements and test. This means we have to deal with three concepts, namely requirements, design artifacts and test cases. In Figure 1, the traceability model, these concepts are captured including their traceability links. The traceability links that need to be captured in this traceability model depend on project specific information needs [10], but also on factors such as schedule and budget [27].

In general, identifying the requirements and test case in the documentation is relatively easy compared to 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. UML. Here, we encounter the well known problem of establishing traceability relations between requirements and design [30]. For this reason we added the dashed line in Figure 1. We hope to improve the requirements to design relation indirectly via the test cases.

4.1.2. LSI and Link Selection

Once LSI created the similarity matrix a choice has to be made if the similarity number is indeed a traceability link or not. In their application of LSI De Lucia *et al.* present several strategies for selecting traceability links. The following are discussed [18]:

- *cut point*; In this strategy we select the top *k* links regardless of the actual value of the similarity measure [3, 21].
- *cut percentage*; In this strategy we select a percentage of the ranked list to be considered as links regardless of the actual value of the similarity measure.

- *constant threshold*; In this strategy we select those links that have a similarity measure greater than c, where c is a constant (a commonly used threshold is 0.7).
- *variable threshold*; In this strategy we select those links that have a similarity measure greater than ε, where ε is calculated according to a percentage of the total similarity measures, e.g. the best 20%.
- *scale threshold*; In this strategy the links are selected according to $\varepsilon = c * MaxSimilarity$, where $0 \leq c \leq 1$ [3].

All strategies have their 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.

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 k best or n% best similarity measures as traceability links.

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.

For this reason we defined two new strategies for link selection:

1. a one dimensional filter on the similarity matrix combining *constant threshold* and *variable threshold*, and
2. a two dimensional filter on the similarity matrix combining *constant threshold* and *variable threshold*.

The first approach defines a constant threshold c and a variable threshold ε depending on the minimum and maximum similarity measure of every requirements vector in the matrix. First all the measurements are compared with the constant threshold to indicate if there is any similarity (and thus traceability links). If all measures are smaller than c there are no links at all. This way we can guarantee a certain level of quality.

If there are measures greater than the constant threshold we take the variable threshold ε for selecting the traceability links. With the variable threshold a similarity interval is defined by the minimum and maximum similarity measures of the requirements vector making it possible to take, e.g., the best 20% of all similarity measures in that vector representation and select them as traceability links. Note that this strategy focuses on the requirements vectors of the similarity matrix. In most cases this strategy is sufficient, but there is one problem with this strategy: it does not consider the other dimension of the similarity matrix (in our case the design vectors and test vectors).

Why is this a problem? Imagine the situation that a design vector has relatively high values for the similarity measures compared to the other design vectors in the matrix. In this case this design artifact returns many traceability links using the first strategy; the similarity measures are higher than the constant threshold c and are also belonging to the interval defined by ε. This is an undesirable situation as one design artifact (or one test case) should not cover all (or most of the) requirements.

This problem is solved using the second strategy. This strategy is basically the same as the first strategy except for the fact it is executed on both dimensions of the similarity matrix. This way it also filters the relatively weak measures of the design and test vectors. In general this should improve the quality of the reconstructed traceability links.

4.2. Assessment Approach

In order to assess the suitability of the reconstructed links, we conduct a qualitative as well as a quantative analysis of the quality 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 [28]. 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 4.3.3.

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

$$recall_i = \frac{|correct_i \cap retrieved_i|}{|correct_i|}$$

$$precision_i = \frac{|correct_i \cap retrieved_i|}{|retrieved_i|}$$

The number of *correct* traceability links are specified in a traceability matrix or known 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. Both

parameters k, c and ε influence the performance indicators recall and precision as will be shown in the case studies.

4.3. Tool Support

4.3.1. Text to Matrix Generator Toolbox

In our case studies we use a Text to Matrix Generator (TMG) toolbox for generating the term-by-document matrix [33]. This toolbox can be used for generating and incrementally modifying term-by-document matrices from text collections.

TMG is designed to perform the preprocessing and filtering of the provided text. The typical IR steps TMG provides are: lexical analysis, stop word elimination, stemming, index-term selection, and index construction.

4.3.2. Trace Reconstructor

We extended the TMG toolbox with a tool named Trace Reconstructor (TR) tool. TR can do the LSI analysis directly after TMG generated the term-by-document matrix. TR supports multiple strategies for selecting traceability links and can directly compare the results of the LSI analysis with the traceability matrix provided by the experts. Furthermore the TR tool calculates the quantitative metrics, recall and precision, from the comparison. The result of the TR tool is emitted as a series of text files containing the relevant traceability matrices.

4.3.3. Trace Explorer

The original documents as well as the recovered traceability links are subsequently input for the Trace Explorer (TE) tool. This tool builds on our earlier work on documentation generation [31], generating a HTML representation of the documents that includes forward and backward links between the documents based on the traceability matrices provided.

Trace Explorer was designed to help us and other researchers in assessing the validity of the reconstructed links. For that reason, it offers support for displaying and comparing the matrix provided by an expert (if available), and the reconstructed one. False positives and false negatives are explicitly marked. Any cell in any matrix is clickable, and leads to a page containing the two documents that are connected through the link. This particularly helped us in seeing which links were (not) reconstructed, and in experimenting with different configurations of our tool set (for example, the LSI parameters) in order to optimize the reconstruction results. This way the TE tool supports the qualitative assessment of the results.

5. Case Studies

In this section we will discuss three case studies where we applied the LSI analysis. The case studies vary in size and context. The first case study, Pacman, is a small case we development 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 is developed by another university as part of an assignment from an external client. The size of that case study is bigger. The last case study is an industrial case study carried out at Philips Applied Technologies. This case study represents a real life project for commercial purposes. In all case studies we will focus on the links between requirements and design, and requirements and test cases.

5.1. Case Study I: Pacman 2.0

In this section we will discuss the results from a lab experiment executed at Delft University of Technology. This experiment is used by students in a lab course for testing object oriented software following Binder's testing approach [5]. The system at hand is a simple version of the well-known Pacman game. An initial implementation for the system is given, and students are expected to extend the test suite according to Binder's patterns, and enhance the system with additional features (which they should test as well). In [16] we already discussed Pacman 1.0. In Pacman 2.0 we primarily changed the content of the text.

5.1.1. Case Configuration

In this case study three main documents are provided: a requirements specification, a design document and a test specification. Along with these documents a traceability matrix is provided, which captures the correct traceability links. These specifications were in plain text already and could be passed directly as input to the TMG toolbox. For the requirements specification the use cases are chosen as requirements entities. The design is event-oriented so the design artifacts are specified according to the events in the system. Finally, every test case is copied as test case entity for our analysis. In total there are 10 requirement entities (use cases), 19 design artifacts and 17 test cases.

As corpus, the collection of all documents was used, including the implementation. This resulted in a corpus of almost 1300 terms (1100 terms without code). Furthermore, for c we took the value 0.7. The other two values k and ε we varied to get an impression of the impact of these values.

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
4.	Pressing the start button re-activates the game

Table 1. UC7 description

5.1.2. Case Results

The recall (R) and precision (P) for this case study are shown in Table 2 for both strategies discussed in Section 4.1.2. The results show a relatively low precision of the links between the requirements and design. This is mainly caused by the many false positives in design artifacts D3.0 and D3.2.1 and use case UC7. Using the second strategy we filter many of these false positives: D3.0 contained 7 false positives using the first strategy and only 2 using the second strategy. For D3.2.1 we realised the same reduction, the 7 false positives where reduced to only 2 using the second strategy, while the correct links remained in the result. Unfortunately, the second strategy did not have any effect on the 10 false positives in use case UC7.

Looking at the links between the requirements and test cases we observed the same result: UC3 and UC7 returned many false positives using the first strategy. The second strategy reduced many (6 out of 7) false positives in UC3. Again, in UC7 none of the 6 false positives were filtered.

The quantitative analysis did not help us to understand this phenomenon so we needed to explore the text. We used the TE tool 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 Table 1) 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 10 false positives. The other similarity measures did not sufficiently decrease for the link selection strategy to filtered them.

		Strategy 1		Strategy 2	
Link type	ε	R	P	R	P
Use case	20%	0.74	0.21	0.63	0.25
to design	30%	0.79	0.18	0.79	0.22
	40%	0.84	0.13	0.84	0.16
	50%	0.84	0.10	0.84	0.14
	60%	1.00	0.10	0.89	0.13
Use case	20%	0.82	0.34	0.82	0.41
to test	30%	0.94	0.30	0.94	0.37
	40%	0.94	0.23	0.94	0.28
	50%	1.00	0.21	1.00	0.24

Table 2. Recall and precision for the reconstructed traceability matrices of Pacman 2.0 with rank-k subspace of 20% and c = 0.7

In this case the constant threshold c did not have any influence on the result. All selected links had a similarity measure above 0.7.

5.2. Case Study II: Calisto

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

In this project a Interface Specification Tool is constructed. This tool is designed to support the ISpec approach; a specification method in the context of component technology [15]. The purpose of the tool is to create Interface Specification Documents as well as exporting these documents to other software components.

5.2.1. Case Configuration

In this case study we again focused on three main documents: a user requirements specification (URD), a software requirements specification (SRS) and an acceptance test plan (ATP). The documents all comply with the equally

		Strategy 1		Strategy 2	
Link type	ε	R	P	R	P
Use case	20%	0.27	0.28	0.22	0.43
to design	30%	0.42	0.21	0.32	0.30
	40%	0.59	0.18	0.49	0.29
	50%	0.63	0.13	0.54	0.19
Use case	20%	0.51	0.41	0.45	0.69
to test	30%	0.71	0.28	0.61	0.50
	40%	0.84	0.20	0.75	0.37
	50%	0.89	0.15	0.81	0.25

Table 3. Recall and precision for the reconstructed traceability matrices of Calisto with rank-k subspace of 20% and c = 0.4

named specifications from the Software Engineering Standard, as set by the European Space Agency (ESA) [1]. However we consider the SRS as a design document as it specifies classes and interfaces.

All requirements have a unique identifier; the user requirements comply to URCARxx and the software requirements to 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.

In this case study we did the analysis including the code and excluding the code. In the first case the corpus consisted of almost 5500 terms, in the second case the corpus consisted of almost 2300 terms. We started with the same values for k, c and ε as in the Pacman case.

5.2.2. Case Results

We again investigated the results of both link selection strategies in Figure 3. We observed that the improvement in precision was even higher than in 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.

In this case study we also 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 0.63 and the recall of test never to exceed 0.94.

5.3. 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 adhoc attempts in two case studies showed that less than 10% of the total requirements can be recovered from the design and test documents. Furthermore, as the system evolves, new requirements are introduced in the system that cannot be traced back to the original requirements specifications.

5.3.1. 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. Furthermore, they are not explicitly related. Thus, no traceability matrix is provided or anything comparable.

In this case it was not very obvious 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 artifact's our choice was not very obvious as well. Every document contains the design of one component or interface in the system. Taking a component as design artifact makes most 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 16 design artifact's.

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 2300 terms representing the engineering domain. Before we executed the LSI analysis on the document set we carried out an simple exploring analysis on the documents.

5.3.2. Preliminary Analysis of Documents

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 [24]. In a first analysis we transformed the Microsoft Word documents to XML format and did some searching on the document set using Xpath expressions [7]. 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 .

5.3.3. 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.

The similarity measures between requirements and test cases showed considerably less good results. 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.

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 did not calculate the recall and precision.

6. Discussion

Link Selection With respect to the link selection strategies we first of all can say that the second strategy performs better than the first strategy.

We have also seen that the strategies cannot always reduce the number of false positives successfully. Best example is use case UC7 in the Pacman case. The 11 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 successfull. They will, regardless of the actual value, simply select only the best x similarity measures (where x is a integer value).

Another point of attention is the constant threshold. In some cases it can be a disadvantage as well. The constant threshold protects you from losing to much quality in your similarity measures. 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 this 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 of no 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 insures 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.

Link Types Futhermore we observe a consequent 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 preformed better. 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. This is still an open issue that remains to be answered.

Research 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 sufficiently many results?
- What documents should be included in the corpus? For example, do we get better requirements–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–design and for requiements–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?

7. 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 defined a new strategy for selecting traceability links from an LSI similarity matrix.
- We provided a tool set for reconstructing traceability links including support for quantitative (TR tool) and qualitative (TE tool) 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 two lines. First, we want to study other links than those between requirements on the one hand and testcases and design decisions on the other, such as those listed in [16]. Furthermore, we are in the process of extending our experimental basis. In particular, we are starting up two new case studies in the area of space engineering and traffic monitoring systems.

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.

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

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

[6] 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.

[7] James Clark and Steve DeRose. Xml path language (xpath) version 1.0. Technical report, W3C, November 1999.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

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

[27] 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.

[28] James Robertson and Suzanne Robertson. Volere requirements specifi cation template. Technical report, Atlantic Systems Guild, 2000.

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

[30] 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.

[31] Arie van Deursen and Tobias Kuipers. Building documentation generators. In *Proc. of the IEEE Int. Conf. on Software Maintenance*, pages 40–49, Washington, DC, USA, 1999. IEEE Computer Society.

[32] 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.

[33] D. Zeimpekis and E. Gallopoulos. Design of a matlab toolbox for term-document matrix generation. In *Proc. of Workshop on Clustering High Dimensional Data and its Applications*, pages 38–48, Newport Beach, California, April 2005.

[34] 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.