# REFORMULATING COMPONENT IDENTIFICATION AS DOCUMENT ANALYSIS PROBLEM
## *Towards Automated Component Procurement*

Hans-Gerhard Gross and Marco Lormans
*Software Engineering Research Group, Delft University of Technology*
*Mekelweg 4, 2628 CD Delft, The Netherlands*

Jun Zhou
*The First Research Institute of the Ministry for Public Security*
*No. 1 South Road of the Capital Gymnasium, 100044 Beijing, China*

Keywords: Latent Semantic Analysis, Component Identification, Component Selection, Procurement Automation.

Abstract: One of the first steps of component procurement is the identification of required component features in large repositories of existing components. On the highest level of abstraction, component requirements as well as component descriptions are usually written in natural language. Therefore, we can reformulate component identification as a text analysis problem and apply latent semantic analysis for automatically identifying suitable existing components in large repositories, based on the descriptions of required component features. In this article, we motivate our choice of this technique for feature identification, describe how it can be applied to feature tracing problems, and discuss the results that we achieved with the application of this technique in a number of case studies.

## 1 INTRODUCTION

The main recent technological advances in component-based development mainly focus on the integration step. This encompasses the syntactic and semantic mapping between components, the development of component wrappers and adapters, and the validation of all pair-wise component interactions (Atkinson et al., 2002; Atkinson et al., 2006; Gross, 2004). Prior to integration, components have to be located in a repository, and evaluated and selected according to non-functional requirements. These activities are typically referred to as component procurement (Gross et al., 2005). Component identification is the first step of procurement, selecting a few candidates out of the huge number of possible components.

In this paper, we assess how well Latent Semantic Analysis (LSA) can be applied for identifying suitable candidate components out of a huge number of possible components in a component repository. LSA is used to automatically induce a specific semantic meaning of given components (Maletic and Valluri, 1999), and, thus, identify candidate components that are matching the semantics of a requirements document.

## 2 RELATED WORK

Component identification is a feature mapping problem that is supported through a number of methodologies and processes.

The Off-The-Shelf-Option (OTSO) (Kontio et al., 1995; Kontio, 1996) contains a definition of the activities involved in component selection, which are described in form of a generic process model. It is based on the definition of evaluation criteria according to reuse goals, and it gives concrete guidelines on how to apply them in a reuse scenario.

The COTS-based Integrated System Development (CISD) model (Tran et al., 1997) can be used to generalize component selection, evaluation, and integration. The model is based on these three phases and provides concrete steps to be taken in each of the phases.

The Procurement-Oriented Requirements Engi-



Gross H., Lormans M. and Zhou J. (2007).
REFORMULATING COMPONENT IDENTIFICATION AS DOCUMENT ANALYSIS PROBLEM - Towards Automated Component Procurement.
In *Proceedings of the Second International Conference on Software and Data Technologies - SE*, pages 111-116
DOI: 10.5220/0001326701110116

neering (PORE) method (Maiden and Cube, 2000) provides guidelines on acquiring customer requirements and selecting components that satisfy these. It offers techniques to discover, acquire, and structure requirements and formulate test cases to be used for assessment.

COTS-Based Requirements Engineering (CRE) (Alves and Castro, 2001) adds non-functional requirements to the component identification process. Its selection criteria proposed comprise functional and non-functional requirements, timing restrictions, cost ratings, and vendor reputation. It can be seen as a method for facilitating requirements engineering in component-based development projects.

These methods provide complete methodological support for component identification, and they facilitate all activities related to finding the right components. They do not, however, provide tools that can support their activities. The approach we propose in this paper is orthogonal to these methodologies. It should be integrated as a tool to support and automate the identification process of the other frameworks, thus reducing the manual effort of feature mapping.

# 3 COMPONENT IDENTIFICATION WITH LSA

The fundamental approach of document analysis or information retrieval techniques is to "match words of queries with words of documents" (Deerwester et al., 1990). We can use the same terminology to describe component identification as "matching words of component requirements" to "words of component descriptions," and reformulate the component identification problem as a document analysis problem that can be solved by retrieval techniques such as LSA (Deerwester et al., 1990).

LSA is an automatic technique for extracting and inferring relations of expected contextual usage of words in documents (Landauer et al., 1998). It takes advantage of implicit higher-order structure in the associations of terms with documents in order to steer the detection of relevant documents (in our case, provided component descriptions in a repository) on the basis of terms in queries (in our case, required component descriptions) (Deerwester et al., 1990).

LSA is based on a terms-by-documents matrix that represents the occurrences of terms in existing documents. The columns of the matrix $A$ correspond to the documents, and the rows correspond to the stemmed and normalized terms. The cells of the matrix contain the number of occurrences of a term in a document. This matrix $A$ is analyzed by singular value decomposition (SVD) to derive the latent semantic structure model (Deerwester et al., 1990), leading to three other matrices $A = T_0 S_0 {D^T}_0$. $T_0$ and ${D^T}_0$ have orthonormal columns, representing the left and right singular vectors, and $S_0$ is diagonal, containing the singular values. If the singular values ($S_0$) are ordered according to size, the first k-largest may be kept and the remaining smaller ones set to zero, leading to a reduced approximate fit with smaller matrices (Lormans and van Deursen, 2006). The product of these new matrices $\hat{A}$ is only approximately equal to $A$ and of rank $k$: $A \approx \hat{A} = TSD^T$. It represents the amended terms-by-documents matrix. The dimension reduction is important for filtering out unimportant details while keeping the essential latent semantic structure intact, and it can be regarded as compressing the same information in a smaller space. Taking the correlation coefficients from this matrix, finally yields the similarity between the documents. High values $[-1..1]$ represent high correlation, low values represent low correlation between documents.

These techniques are initially coming from the software maintenance and reengineering community (DeLucia et al., 2004; DeLucia et al., 2005; Lormans and van Deursen, 2006), where the goal is to establish traceability links between the various development documents. In component-based development, we are facing the same challenges. The semantic concepts described in system-level or component-level requirements must be traced to the corresponding concepts of a component repository. For example, Fig. 1 shows an excerpt of a requirements document and components from a vehicle alarm terminal that can be built into vehicles operated by safety/security services. The system requirements are written in plain text, as well as the component descriptions in the repository. The goal is to use LSA in order to map part of the requirements to existing components in the repository, such that only few candidate components are identified which are likely to implement the required features. This can be done in the following steps (illustrated in Fig. 1 and Table 1) (Landauer et al., 1998; Lormans and van Deursen, 2005; Lormans and van Deursen, 2006): (1) Definition of the traceability model (Fig. 1. Which artifacts take part in the tracing?). (2) Documents are analyzed by LSA, generating a term-by-document matrix (Fig. 1). The columns represent all documents, and the rows represent all relevant terms identified in these documents. The cells represent the occurrence of terms per document (Fig. 1). (3) SVD generates three new matrices (Table 1, $T_0, S_0, {D^T}_0$). (4) Reconstruction of the terms-by-documents matrix $\hat{A}$ out of the reduced SVD-matrices (Fig. 1, $\hat{A}$). This represents the same

Table 1: SVD ($T_0, S_0, D^T{}_0$), dimension reduction ($S_0, k = 3$), matrix reconstruction ($\hat{A}$) and correlation coefficients ($CorrCoef(\hat{A})$).

$T_0 =$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.1408 | 0.1106 | -0.7725 | 0.1213 | -0.5225 | 0.1365 | ... |
| 0.1089 | 0.0763 | -0.5037 | 0.0593 | 0.4066 | -0.5226 | ... |
| 0.0018 | 0.0005 | -0.0221 | 0.0021 | -0.0920 | 0.4767 | ... |
| 0.1441 | -0.7783 | 0.0006 | 0.6109 | 0.0135 | -0.0021 | ... |
| 0.1236 | -0.0218 | -0.2822 | -0.0699 | 0.7286 | 0.4996 | ... |
| 0.4840 | -0.4768 | -0.0381 | -0.7194 | -0.1173 | -0.0647 | ... |
| ... | ... | ... | ... | ... | ... | ... |

$k = 3; S_0 =$

| | | | | | | |
|---|---|---|---|---|---|---|
| 56.6436 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 28.0933 | 0 | 0 | 0 | 0 | ... |
| 0 | 0 | 23.7739 | 0 | 0 | 0 | ... |
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 0 | 0 | 0 | 0 | 0 | ... |
| ... | ... | ... | ... | ... | ... | ... |

$D'_0 =$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.1019 | 0.0128 | -0.5250 | 0.0426 | -0.6482 | 0.4540 | ... |
| 0.1327 | 0.1234 | 0.0983 | 0.1325 | -0.0087 | 0.0662 | ... |
| 0.3909 | 0.2036 | -0.6946 | 0.1027 | 0.2003 | -0.4452 | ... |
| 0.1123 | 0.0207 | -0.2033 | -0.0434 | 0.7069 | 0.5160 | ... |
| 0.5859 | 0.0904 | 0.2642 | -0.2296 | -0.1207 | 0.0388 | ... |
| 0.2354 | -0.1098 | 0.0216 | -0.3555 | 0.0404 | -0.3244 | ... |
| 0.3275 | 0.1271 | 0.1735 | 0.0089 | -0.0304 | 0.1811 | ... |
| 0.1710 | -0.5121 | 0.0596 | 0.5264 | -0.0358 | -0.2704 | ... |
| 0.1731 | -0.3577 | -0.0196 | 0.3084 | 0.1264 | 0.3188 | ... |
| ... | ... | ... | ... | ... | ... | ... |

$\hat{A} =$

| | | | | | | |
|---|---|---|---|---|---|---|
| 8.4033 | 4.2244 | 16.6394 | -2.6378 | 1.5174 | 4.3494 | ... |
| 5.5935 | 2.7813 | 11.3469 | -1.7767 | 1.6187 | 2.3882 | ... |
| 0.2173 | 0.1098 | 0.4169 | -0.0600 | -0.0391 | 0.1866 | ... |
| -2.0765 | -2.5957 | 6.4256 | 2.5521 | 5.1038 | -2.2644 | ... |
| 3.2544 | 1.3799 | 8.3980 | -0.9059 | 3.1904 | -0.1523 | ... |
| 1.3678 | -1.1172 | 16.3386 | 0.5160 | 17.4719 | -11.1565 | ... |
| ... | ... | ... | ... | ... | ... | ... |

$corrcoef(\hat{A}) =$

| | Req | C1 | C2 | C3 | C4 | ... |
|---|---|---|---|---|---|---|
| Req | 1.0000 | 0.9429 | 0.6253 | -0.8798 | 0.0379 | ... |
| C1 | 0.9429 | 1.0000 | 0.3330 | -0.8551 | -0.2225 | ... |
| C2 | 0.6253 | 0.3330 | 1.0000 | -0.5492 | 0.6994 | ... |
| C3 | -0.8798 | -0.8551 | -0.5492 | 1.0000 | -0.2695 | ... |
| C4 | 0.0379 | -0.2225 | 0.6994 | -0.2695 | 1.0000 | ... |
| C5 | 0.1089 | 0.3317 | -0.5509 | 0.1881 | -0.9816 | ... |
| C6 | -0.1619 | -0.3977 | 0.5475 | -0.1094 | 0.9789 | ... |
| C7 | -0.6433 | -0.7199 | -0.2254 | 0.9139 | -0.1930 | ... |
| C8 | 0.6633 | 0.7412 | 0.2295 | -0.9212 | 0.1737 | ... |
| ... | ... | ... | ... | ... | ... | ... |

information as the matrix *A*, though in a smaller subspace, thereby filtering out irrelevant information. For example, the concept "alarm" appeared 13 times in document "Req.", but LSI "estimates" that 8.4033 should be the adapted number of occurrences according to the context usage of the term "alarm" in all other documents (first entry in $\hat{A}$). (5) Calculating the correlation coefficients of the reconstructed matrix yields a new matrix representing the similarity of documents: $CorrCoef(\hat{A})$ in Table 1. (6) Link selection. Which components implement the requirements, or where do we draw the line between interesting components and irrelevant components? (Lormans and van Deursen, 2006) propose several strategies for "ignoring" links.

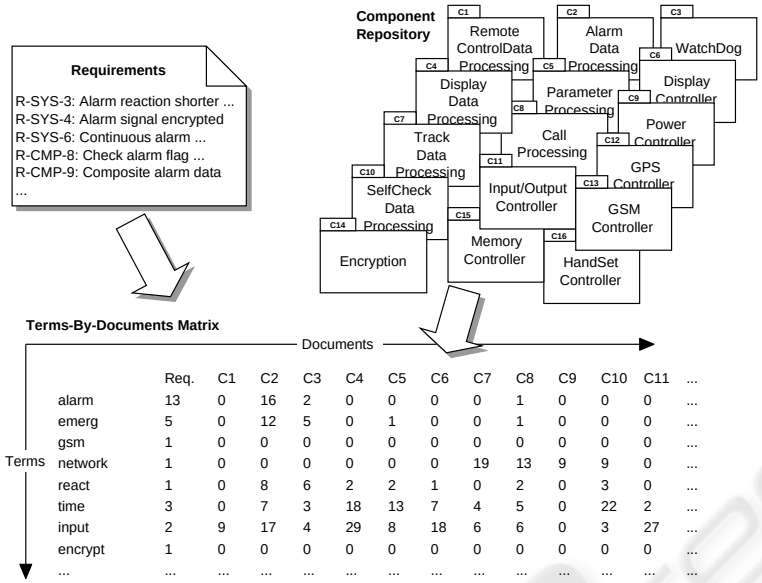
According to the results in our example displayed in Table 1, our analysis method suggests that component C1 appears to be the most suitable candidate (with high probability of 0.9429), but C8 may also be considered (with much lower probability of 0.6633, though).

# 4 LSA EXPERIMENTS

It is important to note that, for assessment of LSA, we require existing systems for which the links between the components are already known. Otherwise, an assessment of the results is difficult. As preprocessing tool, before we apply LSA, we use the TMG Matlab toolbox by (Zeimpekis and Gallopoulos, 2005). It provides a number of functions for stop-word elimination and stemming, and it generates the term-by-document matrices. The other matrix operations are built into Matlab.

**PacMan Case Study.** One of the first case studies in which we applied LSA is a PacMan game used in the Computer Science Bachelor curriculum at Delft University of Technology (Lormans and van Deursen, 2005). The available documentation comprises 10 requirements documents, coming in the form of use case descriptions, 19 documents describing the components of the game and 17 documents with test descriptions, and the Java implementation. The keywords of the programming language do not carry any semantic significance and can be eliminated simply by adding them to the list of stop words. They are then filtered out by TMG. Alternatively, we can use Doxygen to generate documentation out of the source codes and use this as a replacement for the sources.

We included all documents in our analysis, leading to a corpus of some 1200 relevant terms across all documents. We chose the best 20% of all similarity measures as valid links. The value for k (matrix reduction) was varied between 10% and 20% in order to assess the effect of the choices on the selection of links. Best results were achieved with k set to 20%. In that case, LSA was able to identify 16 out of 17 links that had been initially defined by the developer of the program, although LSA found many more links (false positives). This was due to the fact that the use cases described in different documents, leading to different implementations are dealing with similar program events, e.g., one with restarting the game after suspension of the game and one with restarting the game after game over. Both requirements describe similar concepts, and they are, therefore, linked by LSA. The same we found for requirements describing the move of the player and the move of the monsters, or the descriptions about a player bumping into a monster and a monster bumping into the player. All these requirements are describing similar concepts and are, thus, linked by the tool. The link which was not identified by LSA was the description of the GUI of the PacMan and its corresponding test suite. An analysis of the requirements document of the GUI and the




| | Req. | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| alarm | 13 | 0 | 16 | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ... |
| emerg | 5 | 0 | 12 | 5 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... |
| gsm | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| network | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 19 | 13 | 9 | 9 | 0 | ... |
| react | 1 | 0 | 8 | 6 | 2 | 2 | 1 | 0 | 2 | 0 | 3 | 0 | ... |
| time | 3 | 0 | 7 | 3 | 18 | 13 | 7 | 4 | 5 | 0 | 22 | 2 | ... |
| input | 2 | 9 | 17 | 4 | 29 | 8 | 18 | 6 | 6 | 0 | 3 | 27 | ... |
| encrypt | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Figure 1: Analysis and mapping process.

corresponding test document revealed a mismatch of concepts between the two artifacts. Whereas the requirements describe the layout and behavior of the graphical elements of the GUI, the test description is about what a person testing the GUI should look at and which elements should be clicked on. The test description of the GUI was, therefore, linked to most of the other documents, because their concepts appear in the test descriptions. They were not linked to the GUI requirements document, because the concepts described there are different from the ones in the test description, so that LSA does not link them.

**Callisto Case Study.** Callisto is a software engineering tool that can be used to specify component interfaces. We looked at three classes of documents, user requirements specification, component design documents and acceptance test plan, and tried to link those with each other and the implementation. In the experiments including the source code, we ended up with 5500 relevant terms. The parameters of the tool were set to the same values as for the PacMan case. LSA was able to trace 63% of the requirements into the code correctly and 94% of the requirements into the test specification accurately. Linking the requirements to the code produced many false positives. Hence, the low rate of correctly recovered links. It is important to note that the requirements and test descriptions had explicit links through unique identifiers. It was possible to trace the requirements to their respective test documents easily, so that an evaluation of the results for the requirements-to-test tracing was straightforward.

Further, we found that LSA had more difficulties to establish the links between the requirements and the component design documents, than it had for linking the test suites with the requirements. This can be attributed to the fact that many of the design documents contain UML models in the form of pictures capturing many of the essential concepts. The models were not included in our text-based analysis, so that the concepts described there would not make it into the term-by-document matrix. LSA could only consider part of the information contained in the design documents, leading to much weaker links, and, thus, the low value of 68%.

**Philips Case Study.** With Philips Applied Technologies, we carried out a case study in which we tried to link requirements to component descriptions and test descriptions for part of a DVD recorder software. The question was to which extent all requirements agreed in the contract were actually implemented in the end product. Unfortunately, we had no explicit traceability matrix produced by the developers of the system available, so that a final assessment and drawing exact conclusions from the case study was difficult. However, it provided many new insights into the performance of LSA for component feature mapping.

There were requirements on different levels of abstraction available and it was not obvious which of the hierarchy would be the most suitable. For the analysis, we decided to include the first and second highest level of abstraction. Lower-level requirements

seemed to include too many details that could not be traced into the component descriptions. Before we carried out LSA, we tried to find explicit links between the documents, aiming to come up with a manually produced traceability matrix. 20 artifacts were analyzed, all coming in the form of text documents. Preprocessing of the artifacts resulted in 2300 terms in the terms-by-documents matrix.

A noticeable outcome from the experiments was the much higher predicted similarity of concepts between the requirements and the component descriptions, than the similarity between the requirements and the test descriptions produced by LSA. In the other two case studies, it was the other way round. Apparently, the component descriptions were linked well to the corresponding requirements because every component comes equipped with a general high-level description of its functionality that is expressed in an abstract way similar to the high-level requirements. Obviously, the test descriptions were linked poorly to the requirements, because the tests were defined according to the low-level design descriptions of the components which did not correspond to the high-level requirements.

**Discussion of the Results.** Working with the cases presented, provided a lot of insight in how LSA can be applied to linking various types of available documents in a typical software development process. Our primary aim here is to link system level requirements or component level requirements, coming from the decomposition hierarchy of a system, to respective candidate components in a repository by using latent semantic analysis. In the experiments we used all available kinds of documents including high- and low-level requirements, intermediate design documents as well as test descriptions and source code. For LSA it does not matter which documents are belonging to which types of artifacts. It simply tries to guess links between all documents included in an analysis based on an underlying semantic structure inherent in these documents. It is our responsibility to attribute the various types of documents to one distinct entity, i.e., one component. This can be done through copying all relevant information into a single file that represents one component description. LSA will link whatever concepts it finds in other documents that are similar to the concepts of our component description to that particular component. It is, therefore, quite robust with respect to the kind of information provided for each component, as long as it comes in textual form.

In the Callisto case study we had many UML diagrams available, apparently containing essential concepts that were not considered in the analysis. This lead to poor linking of concepts in these documents. A textual description would probably lead to much better results. Graphical notations are more and more being used in industry because people can grasp the essentials of such documents more easily. In the future we will have to look at how we can extract this information automatically and make it available in textual form.

It was interesting to see how well test cases could be traced from requirements. Test cases, especially system level tests and acceptance tests, are usually devised according to the information found in the requirements documents. As a consequence, they are very likely to incorporate similar concepts. After all, they represent implicit links between the implementation (execution of tests) and the outcome of the tests (oracle) coming from high-level requirements or design documents. LSA can make this implicit semantic similarity explicit. Component specifications should, therefore, always come together with their respective test suites according to the tester components described in (Gross, 2004; Gross et al., 2005).

We also observed that low-level implementation-specific test cases could not be traced well to high-level requirements. This was somewhat surprising, since abstraction is the single most important technique for us humans to deal with complex entities, and we expected that we would use the same semantic concepts on higher levels of abstraction that we use on lower levels, though, just getting rid of the details. Apparently, that is not the case, and we have to understand the mechanics of abstraction better.

# 5 SUMMARY AND CONCLUSIONS

In this article, we have introduced and assessed a novel technique for automatically linking requirements to component specification documents through applying latent semantic analysis. Being able to trace concepts that are essential in an application development project to a collection of component descriptions in a repository is the prerequisite for automated component feature detection and analysis. So far, we can only identify the required essential concepts of an application in a component repository, and we can create links in the form of a terms-by-documents matrix to the documents describing the components. However, the links are weighted (coming with a probability), so that we can constrain the number of suitable components to only a few, compared with the potentially huge number of components in a repository.

LSA helps us to identify few relevant components out of a large repository. The experiments that we performed are quite promising with that respect. LSA does not provide support for the next step in component procurement, the assessment of the likely adaptations to be carried out. At this moment we have no answer to this next problem.

For the future, we are planning to perform many more case studies using varying types of documents. It would be interesting to see how more structured documents such as use case descriptions and other templates (Kamsties et al., 2001; Overhage, 2004), that are more and more used in industry, affect LSA. Will such structures improve its performance or will they have a negative effect? The same applies to more formalized documents, such as requirements containing logic and formulae. We have seen already how UML diagrams can inhibit the text-based LSA technique. Is that going to be the same with formal expressions? Another issue that we will look at in the future is how we can extract textual concepts from diagrams that are used in industry (Born et al., 2004).

## REFERENCES


Alves, C. and Castro, J. (2001). Cre: A systematic method for cots selection. In *15th Brazilian Symposium on Software Engineering*.

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wüst, J., and Zettel, J. (2002). *Component-Based Product Line Engineering with UML*. Addison-Wesley.

Atkinson, C., Bunse, C., Gross, H.-G., and Peper, C., editors (2006). *Component-Based Software Development for Embedded Systems*, volume 3778 of *Lecture Notes in Computer Science*. Springer.

Born, M., Schieferdecker, I., Gross, H.-G., and Santos, P. (2004). Model-driven development and testing. In *1st European Workshop on MDA with Emphasis on Industrial Applications*, Enschede, The Netherlands.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 49(6):391–407.

DeLucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2004). Enhancing an artefact management system with traceability recovery features. In *20th IEEE Int. Conf. on Software Maintenance*, pages 306–350. IEEE Computer Society.

DeLucia, A., Fasano, F., Oliveto, R., and Tortora, G. (2005). A traceability recovery tool. In *9th European Conference on Software Maintenance and Reengineering*, pages 32–41. IEEE Computer Society.

Gross, H.-G. (2004). *Component-Based Software Testing with UML*. Springer.

Gross, H.-G., Melideo, M., and Sillitti, A. (2005). Self-certification and trust in component procurement. *Science of Computer Programming*, 56(1–2):141–156.

Kamsties, E., von Knethen, A., and Paech, B. (2001). Structure of quasar requirements documents. Technical report, Fraunhofer IESE, Kaiserslautern.

Kontio, J. (1996). A case study in applying a systematic method for cots selection. In *18th Intl Conference on Software Engineering (ISCE-1996)*, pages 201–209, Berlin.

Kontio, J., Chen, S., and Limperos, K. (1995). A cots selection method and experiences of its use. In *20th Annual Software Engineering Workshop*, Greenbelt Maryland. NASA Goddard Space Flight Center.

Landauer, T., Folz, P., and Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284.

Lormans, M. and van Deursen, A. (2005). Reconstructing requirements coverage views from design and test using traceability recovery via lsi. In *3rd Intl. Workshop on Traceability in Emerging Forms of Software Engineering*, pages 37–45, Long Beach.

Lormans, M. and van Deursen, A. (2006). Can lsi help reconstructing requirements traceability in design and test? In *10th IEEE Conference on Software Maintenance and Reengineering*, Bari, Italy.

Maiden, N. and Cube, C. (2000). Cots software selection: The need to make tradeoffs between system requirements, architecture and cots components. In *COTS workshop. Continuing Collaborations for Successful COTS Development*.

Maletic, J. and Valluri, N. (1999). Automatic software clustering via latent semantic analysis. In *14th Intl Conference on Automated Software Engineering*, page 251.

Overhage, S. (2004). *Object-Oriented and Internet-Based Technologies, Weske and Liggesmeyer (Eds)*, volume 3263 of *Lecture Notes in Computer Science*, chapter UnSCom: A Standardized Framework for the Specification of Software Components. Springer, Heidelberg.

Tran, V., Lui, D., and Hummel, B. (1997). Component-based systems development, challenges and lessons learned. In *8th International Workshop on Software Technology and Engineering Practice*, pages 452–462.

Zeimpekis, D. and Gallopoulos, E. (2005). Design of a matlab toolbox term-document matrix generation. Technical report, High-Performance Information Systems Laboratory, University of Patras. http://scgroup.hpclab.ceid.upatras.gr/scgroup/Projects/TMG.