Supporting Architectural Restructuring by Analyzing Feature Models

Ilian Pashov, Matthias Riebisch, Ilka Philippow
Technical University Ilmenau, Germany
{Ilian.Pashov Matthias.Riebisch Ilka.Philippow}@TU-Ilmenau.DE

Abstract

In order to lower the risk, reengineering projects aim at high reuse rates. Therefore, tasks like architectural restructuring have to be performed in a way that developed new system architectures allow reuse of all valuable legacy systems' parts with minimal changes. During architectural restructuring there are two major types of modification: detection of architecture disproportions and their refactoring and detection of redundancies and their fusion. In this paper we introduce a method for applying domain knowledge for supporting these restructuring steps. The method operates on feature models. Words and terms of features and of architectural documents are analyzed by cluster analysis, information retrieval and metrics techniques. In this way, the method joins the approaches of feature analysis and of enhancing reengineering with domain knowledge by applying feature models for structuring the domain knowledge. The method results in clues and hints for the development of a new architecture. It provides an effective addition to the conventional software architecture design methods.

The method was developed and applied in an industrial reengineering project within image processing domain. It has been proved to be applicable to large and complex systems even in case of heavy monolithic parts. We use examples from this project to illustrate the method.

Key Words

Architecture Reconstructing, Architecture Recovery, Reengineering, Program Comprehension, Feature Modeling, Information Retrieval

1 Introduction

Architectural restructuring is an important task within reengineering projects. In order to make a legacy system conform to new domain requirements, often it needs significant architectural changes. Nevertheless it is rare the case that a brand new system architecture is independently developed. More often the newly developed architecture is a restructured version of the current one, which eliminates its major disadvantages. Architectural restructuring includes two big activities: architecture recovery and development of a new architecture considering the valuable parts of the legacy one.

Architectural recovery states the basis for a new system architecture. The activity aims to recover information about the current system architecture, which was lost or became outdated over the years. The results of the activity are the input for new architecture design. Architectural recovery requires good program comprehension and reliable information sources. In the case of little and outdated system documentation the most reliable information can be derived from the source code. However, the lack of abstract information hinders the comprehension. There is often an abstraction gap between source code, system documentation and architecture. To bridge this gap domain information is applied with great success, i.e. in form of feature models [22]. Features present customer valuable properties of systems. Feature models structure features regarding their influence over the system architecture. Feature models can be analyzed and the results of this analyses can support system architects in restructuring of the current architecture and by new system design.

The development of new system architecture is probably the key task of reengineering projects. Considering the results from architecture recovery, the newly developed architecture has to make a system conform to the latest domain requirements and at the same time to allow integration and reuse of all significant parts of the legacy code. If the last is not true the reengineering projects risk increases undesirable high and the success of the projects can be endangered [15]. The new system architecture has to deal with the disadvantages of the legacy one as well. It has to consider and eliminate the architecture disproportions and the redundancies. There is a number of methods based on source code analyses, which can support these tasks: [1], [3], [16], [19], [18] and [7]. All of them can be applied with more or less success, but since they work on the lowest design level they tend to omit high level information. These methods use domain knowledge in form of experts opinion, but they lack of analyzing high level structured domain information.

In this paper we present an approach, which uses domain information analyses to assist detection of architecture disproportions and redundancies within a legacy system. Our method takes the input of system features structured in a corresponding model, analyses them and produces a set of clues and hints showing the above mentioned architecture problems and their possible solutions. The method results provide additional information about clusters of features and possible separation of concerns within a software system. The approach is based on text analyses information retrieval techniques. Names of features are used as information source and related requirements are considered in case of insufficient information. A prerequisite for the method is the existence of feature model and traceability links between features and system architecture. These can be established using one of the approaches presented in [22], [8] or [24].

For illustration of the proposed techniques we use examples from an industrial reengineering project within image processing domain. Most of the presented ideas have been developed during the work on this project and are influenced from real problems, which we encountered during our work. Although some of our examples may have the specifics of the studied system, we believe that the presented methodology can have general application for reverse engineering tasks.

2 Method for Supporting Architecture Restructuring

2.1 Feature Modeling

Before we start to describe our approach, a short introduction to feature modeling is required, as this is the utilized key technology.

Feature modeling is a method originally developed for structuring domain properties from customer‘s point of view. Features are often referred as customer valuable properties of a system, which can be selected in a product configuration. Feature model (Figure 1) is a hierarchical structure of features, which shows additionally grouping and constraint relations between features. In a feature model, each feature can be marked as optional or mandatory. In this way, feature modeling provides means for managing the variability of systems in a problem domain.

Feature modeling was introduced from the Feature-Oriented Domain Analysis (FODA) methodology [17] and further developed from a number of approaches, for example: [12], [6], [23], etc. Due to the ability of the feature models to present important information in a structured way, which is also close to the structure of the architecture components in a system, feature modeling was recognized in [22] as eligible technology for supporting program comprehension and software architecture recovery. In this previous work, we showed how a feature model can be the necessary artifact, which bridges the abstraction gap between source code, system documentation, architecture and requirements. We presented a method, which collects domain information in a feature model through reverse engineering of system documentation and through performing expert interviews. Later, the established feature model was utilized for generating and verifying hypotheses about architecture elements, thus assisting the reverse engineering work by providing structured high level design information. During the architecture recovery were also established traceability links between features and architecture elements. For example on Figure 2 are shown sample feature model, architecture components and their traceability links from an image processing system. The traceability links show which feature in which architecture component is implemented.

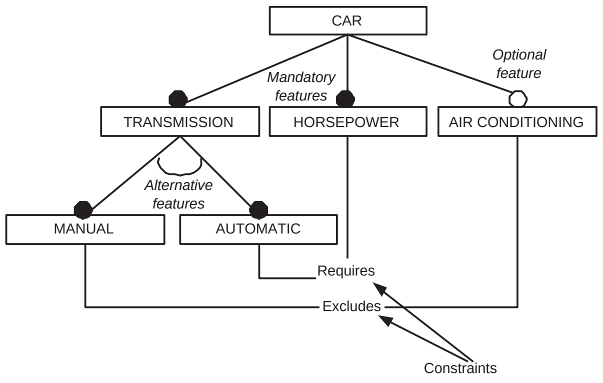


Figure 1. Sample Car Feature Model - adapted from [17]

2.2 Overall Description

On the basis of the method presented in [22] we have developed an approach for assisting architecture restructuring. In this approach we use as input the feature model and the traceability links provided from the previous method. Over the feature model we run analyses, which provide useful hints and clues for eliminating design problems by architecture restructuring activities.

The method (Figure 3[1]) starts with recovery of the current architecture and establishment of a feature model of

[1]For the diagrams on Figure 3 and Figure 5 are used the Gane-Sarson DFD-Notations [11].

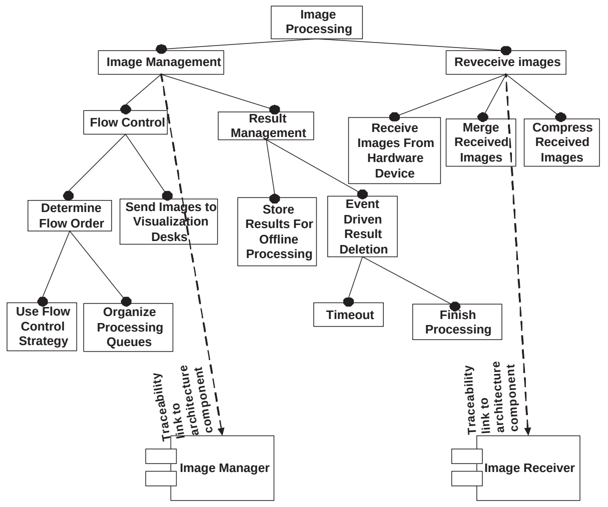
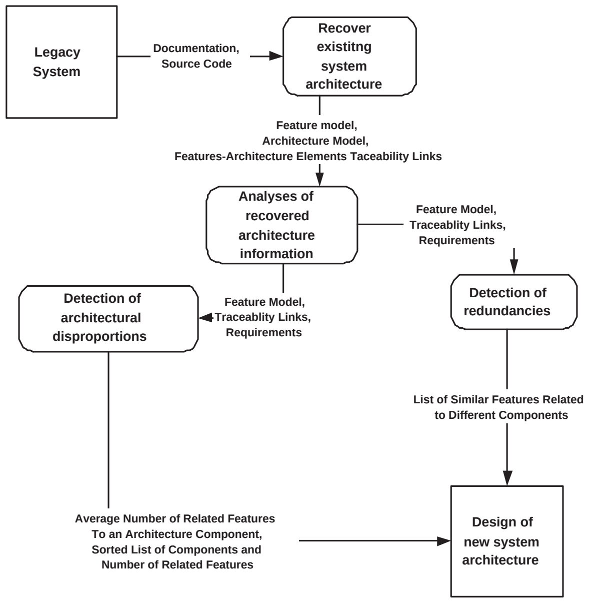


Image Processing
Image Management
Reveceive images
Flow Control
Result Management
Receive Images From Hardware Device
Merge Received Images
Compress Received Images
Determine Flow Order
Send Images to Visualization Desks
Store Results For Offline Processing
Event Driven Result Deletion
Use Flow Control Strategy
Organize Processing Queues
Timeout
Finish Processing
Traceability link to architecture component
Traceability link to architecture component
Image Manager
Image Receiver

Figure 2. Sample Feature Model and Traceability Links Between Features and Components



Legacy System
Documentation, Source Code
Recover existitng system architecture
Feature model, Architecture Model, Features-Architecture Elements Taceability Links
Analyses of recovered architecture information
Feature Model, Traceablity Links, Requirements
Detection of redundancies
Feature Model, Traceablity Links, Requirements
Detection of architectural disproportions
List of Similar Features Related to Different Components
Average Number of Related Features To an Architecture Component, Sorted List of Components and Number of Related Features
Design of new system architecture

Figure 3. Supporting Architectural Restructuring - Activities and Data Flow

the reengineered system. For these tasks can be applied for example the [22] approach. If these artifacts are present the step can be omitted. Traceability links between features and architecture components are also required. In case they are missing, their establishment should be considered as an activity as well. On the next step, the feature model and the recovered architecture information are analyzed in order to detect architecture disproportions and redundancies within system architecture. These analyses are detailed described in sections 2.3 and 2.4. The results of the analyses are used as hints and clues from system architects by designing a new system architecture, which has to take into account the old design elements of the reengineered system (section 2.5).

2.3 Detection of Architectural Disproportions

Evolution of software systems often leads to disproportional growing of parts, which in the initial system architecture have been designed as equally complex. As a result, after a time the complexity balance between architecture components is completely lost and monoliths have appeared. The monoliths have lower maintainability and breaking them is a common task in reengineering projects.

Usually, detection of monoliths is not a difficult tasks since most of them are obvious and the related problems have been observed a long time from the system experts. Nevertheless, even the best experts miss sometimes important facts and on the other hand need supporting arguments for their opinion. A methodology for detection of architecture disproportions within software system architectures will be of good use for reverse engineers and software architects. It will show them how the features are distributed over the system and will hint potential monoliths.

We propose a statistical method, which analyses system features and related architecture components and detects disproportions between the components. In brief the method:

- Calculates the number of features related to an architecture component.
- Calculates the average number of features implemented in all architecture components.
- Positions the architecture components according to the diversion on the number of related features towards the average number of implemented features.

All calculations are made over the feature model of the studied system and the traceability links to the architecture components. Following the traceability links, it is possible to count the number of features related to an architecture component. The average number of features is calculated as an average value of the number of all features and the number of all components. Table 1 presents sample results of the method.

As it is visible from Table 1, there is a significant difference in the number of related features between the components at the bottom and at the top of the table. The components at the top of the table, especially the IM (Image Manager) concentrate a lot of functionality and one can say

Component	**Features Per Component**
IM (Image Manager)	53
FO (Flow Observer)	21
VD (Visualisation Desk)	18
UII (User Interface Interceptor)	18
DISP (Dispatcher)	17
OCR (Optical Character Recognizer)	*16*
EIMSGW (External Information Management System Gateway)	*12*
II (Image Injector)	*10*
ISIF (Image Source Interface)	9
ICIF (Image Consumer Interface)	9
SC (Statistics Collector)	7
IDB (Image Database)	5
XVI (External Video Interface)	5
IR (Image Receiver)	4
ISDB (Image Streams Database)	3
TD (Test Desk)	3

Table 1. Features per component classification example

that they have become monoliths. On the other hand the components at the bottom of the table are quite simple, although they are classified on the same architecture level as the other ones. The components in the middle of the table (the italic font) implement number of features close to the average for the system (13 features per component[2]) and can be considered well balanced.

2.4 Detection of Redundancies

Another common problem between systems being subject of reengineering are the redundancies. It is known that a lot of code is developed using ”copy paste” techniques and the literature is rich of approaches which deal with this problem, known as clone detection. Unfortunately, ”copy paste” development is not the only reason leading to redundant development. Many problems come from the system design itself. Due to a number of reasons including system architecture decay, a lot of requirements are solved more than once. Implementation for one and the same feature can be found more than once in different architecture components. Sometimes the feature has slightly or even much different name in the different architecture components, but the experts claim that it is one and the same.

Since the clone detection techniques work on source code level and omit high level design information it is hard using them to detect redundancies caused by architecture design problems. In our opinion, the features can help the detection of such problems. Features stay between architecture and system requirements and are influenced from both. Features are described with terms from the domain vocabulary and concentrate domain information, which allows application of information retrieval algorithms over them. Analyses of features and their traceability links to architecture components can show redundancies caused from the system design.

We propose a design redundancies detection method, which does in brief:

- Application of text analyses information retrieval techniques over a database of feature names in order to detect and order the main system concerns. We perform two types of analyses:
  - Words frequency analysis.
  - Words categorization analysis.
- Clustering of features according to text analyses results.
- Examining features classified in a same cluster for similarities.

Figure 4 shows an example of possible redundancies, which we detected in our studied system. There was a set of five features (on the left side) implemented in different components (on the right side), which finally realized the same tasks (on the top). All of them were doing protocol mapping from external protocols to an internal one, but there were several different implementations of the mapping functionality and no unified interface to the native protocol, although for all components it was the same. Due to the redundant design there also were performance problems, since the different components were running as separate processes, which is proved to be more resource consuming that multi threaded architecture for example.

In order to apply information retrieval algorithms there is a need to prepare the above mentioned database with information, which can be analyzed. We put in this database the names of the features describing a feature model and over the database we run text analyses. The information retrieval analyses need a thesaurus as well. In our case we used the thesaurus provided from the used information retrieval tool[3].

Since normally feature names are relatively short and the contained text information could be insufficient for information retrieval, additionally can be added to the information database and analyzed the related to the features requirements. Nevertheless, even without taking the requirements into account we can argue about the correctness of

[2]+/- 20% Project specific tolerance

[3]We have used the PolyAnalyst^{TM} (www.megaputer.com) tool for our information retrieval analyses.

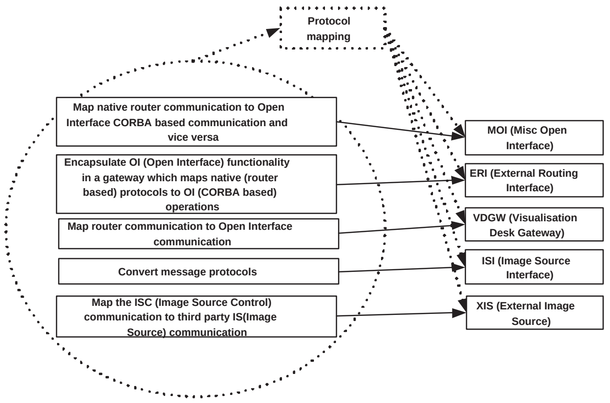

Protocol mapping
Map native router communication to Open Interface CORBA based communication and vice versa
Encapsulate OI (Open Interface) functionality in a gateway which maps native (router based) protocols to OI (CORBA based) operations
Map router communication to Open Interface communication
Convert message protocols
Map the ISC (Image Source Control) communication to third party IS(Image Source) communication
MOI (Misc Open Interface)
ERI (External Routing Interface)
VDGW (Visualisation Desk Gateway)
ISI (Image Source Interface)
XIS (External Image Source)

Figure 4. Possible Redundancies Detected in an Image Processing System

the later described information retrieval analyses due to the following reasons:

- Feature names explicitly use domain technology, which concentrates significant domain information.
- Additionally, feature models are created considering experts opinions and are approved from them (see [22]), thus the correctness of the used terminology is guaranteed.

With **words frequency analysis** we try to find out and classify the most frequently used words within the studied database. The result of the analysis is a report, which shows in how many records (in our case features) these words are met and what percentage they are of the whole number of studied records. As a rule, in the results are presented only words, which are met in more than a certain percent of the records. Table 2 shows sample analysis results from our studied image processing system. The barrier is occurrence in at least 2.214% of the records [4].

Word	Frequency
Image	20(21.74%)
Processing	17(18.48%)
statistics	6(6.522%)
device	6(6.522%)
control	6(6.522%)
result	5(5.435%)
resource	4(4.348%)

Table 2. Words Frequency Analysis Example

[4]This number was chosen as recommended from the used information retrieval tool. In general, the barrier occurrence is dynamically calculated and depends on the number of analyzed different words and the used thesaurus.

With **words categorization analysis** we aim at identifying categories and sub-categories of context-related words. The analysis searches for classes and subclasses of words depending on their relation in a word construction, for example sentence or in our case feature name. The result of the analysis is a categorization tree, which presents the identified word categories and sub-categories. Table 3 shows categorization tree example from our studied system. In brackets in the categorization tree are given the number of records (features), where the categorized word is met.

image(20)	
	convert(4)
	compress(1)
	TIFF(1)
	source(2)
	format(2)
	others
statistic(3)	
	report(3)
	counter(2)
	record(1)

Table 3. Words Categorization Analysis Example - Categorization Tree

The results from the above described analyses are used for **clustering of features** and **finding of duplicated and similar features**. We have developed a semi-automated process for clustering[5] features. The process is manually driven from experts, but we have developed a tool which supports them. The tool helps the experts to navigate through features and define feature clusters, but the assignment of features to different clusters is manually done from the experts. Figure 5 shows the activities and the data flow of the feature clustering process.

The results of the information retrieval analyses are used for definition of feature groups (clusters of features). Each feature group is described with an unique name and a list of keywords. The words from the word frequency analysis determine the names of the groups. The words from the word categorization tree determine the list of keywords for the groups. Additionally the experts can extend and tune the list of keywords and the groups names. They can also define groups which were not detected from the information retrieval analyses. Figure 6 shows an example form for editing feature groups.

Once the feature groups are defined features have to be assigned to them. For this purpose all features are listed in a feature group assignment form (Figure 7). The experts filter

[5]Features describing similar or logically related functionality are considered from one and the same cluster.

iteratively the listed features. The applied filter is a logical union of keywords describing a feature group (e.g. ”Load or Save or Store”). After the filter is applied the resulting records are explored and the experts assign the appropriate features to the group. The filter and the keywords can be refined as well after the filtering results are considered. In case of refinement, the new filter has to be applied. If the new filtering results are satisfying (the listed features belong to the same group) all appropriate features are assigned to the group. The filtering and assignment steps are repeated for each defined group.

For better grouping of features it can be necessary to break some groups into subgroups. The subgroups are defined as normal groups and considered as such in the features assignment form. The difference is that the original group is considered as a super group for these features. This technique allows unlimited hierarchy of feature groups to be reached. When the grouping is finished the resulting feature groups will present clusters of features, which have the same concern.

Table 4 shows some sample groups, which we identified in our studied system. Table 5 lists a set of sample features, which we classified to an ”Image TIFF Operations” group. Figure 8 is a snapshot of a report showing the feature grouping results (the established feature clusters).

Feature Group	Keywords
Input Output	Load, Save, Store, Delete, File, Container, Erase, Disk
Image TIFF operations	Image, TIFF, Convert, Compress, Store
Statistics	Statistic, Counter, Report

Table 4. Sample Feature Groups and The Corresponding Keywords

Comp.	Feature	Group	Super Group
II	Convert raw image data into standard format (TIFF)	Image TIFF operations	Image
ISIF	Compress images (TIFF FAX4/JPEG)	Image TIFF operations	Image
IR	Compress received images (TIFF FAX4)	Image TIFF operations	Image
IDB	Store images in TIFF format	Image TIFF operations	Image

Table 5. Features Classified To "Image TIFF Operations" Group

The features classified in one and the same cluster can be **examined for similarities**. In case similar features are found the traceability links will show in which architecture components these features are implemented, which will point the redundant design. Figure 9 shows the form, which helps the similarities examination. The form allows definition of new features as well, which cover the similar ones. Table 6 shows example of features from the ”Image TIFF Operations” group, which we considered similar and the resulting new feature. Three of the listed four features had the same tasks and could be considered identical.

Comp.	Feature	Resulting Feature
II	Convert raw image data into standard format (TIFF)	Compress Images In TIFF format
ISIF	Compress images (TIFF FAX4/JPEG)	Compress Images In TIFF format
IR	Compress received images (TIFF FAX4)	Compress Images In TIFF format
IDB	Store images in TIFF format	Compress Images In TIFF format

Table 6. Similar Features From The "Image TIFF Operations" Group

2.5 Deduction of Clues and Hints for Architecture Development

The results from the above described analyses provide a set of clues and hint, which if considered can support system architects in architecture restructuring. Namely:

- A new designed system architecture should consider and if necessary should break the monoliths pointed by the architecture disproportions analysis and should remove the problems detected by the redundancies analysis.
- The number of related and implemented features can be considered as a complexity metric for architecture components. Although such metric is very subjective and not an exhaustive one it gives quick and good ”first look” orientation for the complexity of architecture components. Keeping an eye on the number of features covered from the designed components helps the architects to avoid making preconditions for monolithic development.
- The established feature clusters during the redundancies analysis can put the development on the right track finding the right architecture components. Our experience shows that these results are not sufficient for complete components design, but the information provided from the established feature clusters is good clue for the architecture designers.

- The redundancies analysis shows the overlapping functionality within the system. It can help the architects to define better abstractions and to avoid the double work in the new system design.

3 Related Work

A very detailed and comprehensive overview about existing research approaches and open questions in the field of restructuring and refactoring is given in [20] They point out that there is an urgent need for techniques to support the reducing of software complexity by incrementally improving the internal software structure. They also consider the necessity to apply refactorings at a higher abstraction level than source code e.g. applying on UML design models similar to [5], [27] or to use Gamma Design Patterns for a high level program structure description. Approaches that are focused on the integration of domain knowledge in form of features that are derived from existing architectures or located in the source code and enhanced with current domain requirements for supporting design decisions are [8] and [24].

Before carrying out refactoring activities based on feature models the recovered and described architecture must be analyzed in consideration of architectural disproportions and redundancies. Therefore metrics and clone analyzing techniques can be useful integrated into the architectural restructuring process.

In [9] an approach is presented that provides design principles and rules based on a factor-criteria-metrics-model. Several components of an object oriented system can be analyzed to point out the conformity to specific design goals and to derivate decisions for the software stability improvement. This technique can be used in addition to the proposed method.

Decomposing software systems into modules leads to benefits for software flexibility and comprehensibility. In [13] a tool is introduced that assists refactoring of source code in order to achieve a proper package structure. Therefore they use a metric for evaluating the quality of package structure. The metric is defined as the weight of all desirable dependencies in all packages divided by the tool weight of the dependencies in all packages and provides a way to quickly evaluate the internal quality of large software products based on their source code. Similar to the work in [26], metrics are used in our method to detect a need for refactoring a given software system. Statistical Techniques can also be used to provide empirical measurement on the practical use of refactoring.

Clone detection is necessary for finding of redundancies. For lowering of software complexity particular software clones can be removed. In [4] a comprehensive comparison of different clone detection techniques is presented and evaluated by consideration of the achieved values for the recall and precision metrics. Most of the techniques operate on source code level only.

For detection of clones several techniques have been investigated considering their usefulness for supporting refactoring activities. Some of them are based on a full source code text view, other focus on whole block sequences using metrics or pattern matching techniques. All approaches provide useful information about inherent clones but in the most cases the information is not enough for clone based refactoring.

In [14] the Gemini environment is described that can be used for analyzing the code clones and for modifying them e.g. for reducing the clone pairs and clone classes by using a so called Code Clone Shaper. In [21] the relation between software reliability and maintainability in connection with software code clones is shown. It is explained that modules having code clones can be more reliable on average than modules having non code clone. But modules having very large code clones are getting less reliable and are less maintainable. An approach for clone-analysis which focuses on the extraction of clone differences and their interpretation in terms of programming language entities and on the study of contextual dependencies is introduced in [2]. This approach supports the computer-aided refactoring process and it is a good extension to our approach. Refactoring decisions can be done based on provided general information and special characteristics of selected clone clusters.

Information Retrieval offers various techniques for evaluating natural language documents [25]. The basic principle of concluding the importance of a concept in a particular environment from its frequency is applied here. Features and requirements documents are evaluated by analyzing texts in relation to other documents of a domain.

There are several methodologies for describing and structuring domain knowledge. The ASIUM system that is described in [10] supports the acquisition of semantic knowledge from texts based on a conceptual clustering method.

4 Conclusion

The increased demand for reengineering of software systems in the last decades has provoked research in the corresponding direction. A lot of technologies have been developed to help overcoming the related problems. Some of the approaches are automated, some semi-automated or fully manual. Finally the practice shows that still reengineering projects require a lot of manual work and expert help. Every system being subject of reengineering has its peculiarities and only people who are deep into it and its problems can restructure it, redesign it and make it conform to the latest software technologies and to the present user needs. Reengineering methods support the work of these people. They help them recover information, which is lost, outdated or has never been documented. The results of the reengineering approaches are often considered as clues and hints from the system experts by performing their redesign activities.

In this paper we showed how system experts can get some good clues and hints about architecture restructuring activities from problem domain information gathered in a feature model of a reengineered system. We showed how analyses performed over feature models and application of information retrieval techniques can hint system experts existing design problems like unbalanced architecture and design redundancies. Additionally the proposed analyses provide hints about overcoming design problems and about possible separation of system tasks over architecture components.

The proposed approach is driven from real problems observed in an industrial project. Being such one it carries the specifics of the studied system, but in our opinion it has the potential to be successfully applied in other reengineering projects.

5 Future Work

We plan to continue our work in researching the possibilities for application of features and feature modeling in reengineering projects. Feature models give a very comprehensive structure of the domain knowledge and contain architecture information as well. Due to these characteristics of the feature models we see a good potential for their application in reengineering projects. Up to now we found out how feature modeling can be applied for program comprehension and architecture recovery and how analyses of feature models can produce a set of clues and hints, which support architecture restructuring and new design. Now we intend to direct our research to find out how features and feature modeling can help overcoming system architecture decay and obsolescence of the whole system. We would like to find the right fundamentals, which will move away as much as possible the time point when the next reengineering will be required.

References

[1] S. Baker.
On finding duplication and near-duplication in large software systems.
In Newcomb P. Wills, L. and Chikofsky E., editors, *Processdings of the Second Working Conference on Reverse Engineering*, pages 86–95. IEEE Computer Society Press, Juli 1995.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lage, and K. Kontogiannis.
Advanced clone-analysis to support object-oriented system refactoring.
In *Processdings of the 7th Working Congress on Reverse Engineering (WCRE 2000)*, pages 98–107, Brisbane, 2000. Computer Society Press.

[3] I. Baxter, A. Yahin, L. Moura, M. Santanna, and L. Bier.
Clone detection using abstract syntax trees.
In *Processdings of the International Conference on Software Maintenance, 1998*, 1998.

[4] S. Bellon.
Vergleich von techniken zur erkennung duplizierten quellcodes.
Master's thesis, Institute for Informatics, University of Stuttgart, 2002.

[5] M. Boger, T. Sturm, and P. Fragemann.
Refactoring browser for uml.
In *Processdings of the 3rd Intl Conf. on eXtreme Programming and Flexible Processes in Software Engineering*, pages 77–81, Alghero Italy, 2002.

[6] K. Czarnetcki and U. Eisenecker.
Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models.
Addison Wesley, 2000.

[7] S. Ducasse, M. Rieger, and S. Demeyer.
A language independent approach for detecting duplicated code.
In *Processdings of the International Conference on Software Maintenance (ICSM99)*, 1999.

[8] T. Eisenbarth, R. Koschke, and Simon D.
Aiding program comprehension by static and dynamic feature analysis.
In *Processdings of the International Conference on Software Maintenance*. IEEE Computer Society Press, November 2001.

[9] K. Erni and C. Lewerentz.
Applying design-metrics to object-oriented frameworks.
In *Processdings of the 3rd International Software Metrics Symposium*, pages 64 – 74, Berlin, Germany, March 1996. IEEE Computer Society Press.

[10] D. Faure and C. Ndellec.
A corpus-based conceptual clustering method for verb frames and ontology.
In *Processdings of the LREC workshop on Adapting lexical and corpus resources to sublanguages and applications*, Granada, Spain, Mai 1998.

[11] C. Gane and T. Sarson.
Structured Systems Analysis.
Prentice-Hall, 1979.

[12] M. Griss, J. Favaro, and M. dAlessandro.
Integrating feature modeling with the rseb.
In *Processdings of the Fifth International Conference on Software Reuse*, pages 76–85, Victoria, Canada, June 1998. IEEE Computer Society Press.
see http://www.intecs.it.

[13] E. Hautus.
Improving java software through package structure analysis.
In M. Blaha F. Balmas and S. Rugaber, editors, *Processdings of the WCRE'99 (6th Working Conference on Reverse Engineering)*, Oct. 1999.

[14] Y. Higo, Y. Ueda, K. Kamira, S. Kusumoto, and K. Inoue.
On software maintenance process improvement based on code clone analysis.
In *Processdings of the 4th International Conference, PROFES 2002*, pages 185–197, Rovaniemi, Finland, December 2002. LNCS 2559, Springer.

[15] L. Hyatt and L. Rosenberg.
A software quality model and metrics for identifying project risks and assessing software quality.
In *Processdings of the 8th Annual Software Technology Conference*, 1996.

[16] T. Kamiya, S. Kusumoto, and K. Inoue.
Ccfinder: A multi-linguistic token-based code clone detection system for large scale source code.
IEEE Trans. Software Engineering, 28:654–670, 7 2002.

[17] Kyo C. Kang, G. Sholom, J. A. Cohen, W.E. Hess, A. Novak, and S. Peterson.
Feature-oriented domain analysis (foda): Feasibility study.
Technical report, CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[18] K. Kontogiannis, R. DeMori, M. Bernstein, M. Galler, and E. Merlo.
Pattern matching for design concept localization.
In *Processdings of the Second Working Conference on Reverse Engineering*, pages 96–103, Toronto, Ontario, July 1995. IEEE Computer Society Press.

[19] J. Krinke.
Identifying similar code with program dependence graphs.
In *Processdings of the Eighth Working Conference On Reverse Engineering (WCRE01)*, 2001.

[20] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, and P. Van Gorp.
Refactoring: Current research and future trends.
In *Processdings of the ETAPS workshop LDTA 2003*, 2003.

[21] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto.
Software quality analysis by code clones in industrial legacy software.
Technical report, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-TR2001007, September 2001.

[22] I. Pashov and M. Riebisch.
Using feature modeling for program comprehension and software architecture recovery.
In *Processdings of the 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03)*, Huntsville Alabama, USA, April 2003. IEEE Computer Society.

[23] M. Riebisch, K. Boellert, D. Streitferdt, and I. Philippow.
Extending feature diagrams with uml multiplicities.
In *Processdings of the Integrated Design and Process Technology (IDPT) 2002*, pages 1–7, 2002.

[24] C. Riva and J. V. Rodriguez.
Combining static and dynamic views for architecture reconstruction.
In *Processdings of the Sixth European Conference on Software Maintenance and Reengineering*, Budapest, March 2002.

[25] G. Salton and M. J. McGill.
Introduction to Modern Information Retrieval.
McGraw-Hill, 1983.

[26] F. Simon, F. Steinbrckner, and C. Lewerentz.
Metrics based refactoring.
In *Processdings of the European Conf. Software Maintenance and Reengineering*, pages 30–38, 2001.

[27] G. Suny, D. Pollet, Y. LeTraon, and J.-M. Jzquel.
Refactoring uml models.
In *Processdings of the UML2001*, pages 134–138. LNCS 2185, Springer, 2001.

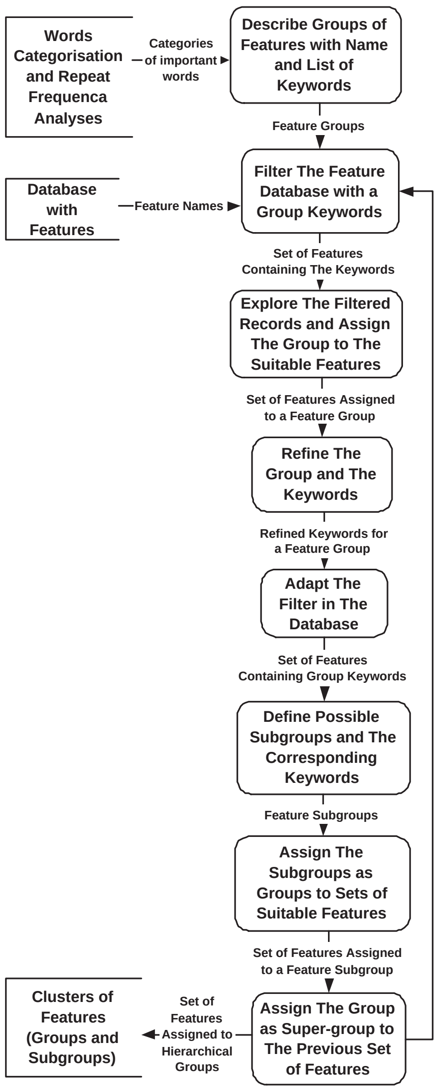
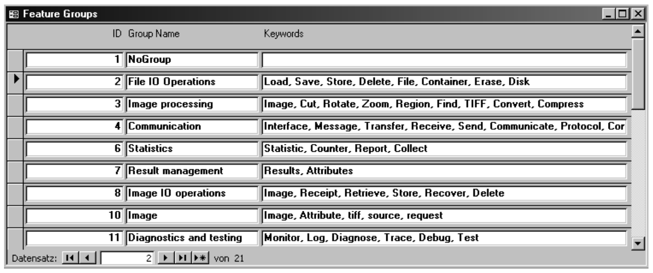

Words Categorisation and Repeat Frequenca Analyses
Categories of important words
Describe Groups of Features with Name and List of Keywords
Feature Groups
Database with Features
Feature Names
Filter The Feature Database with a Group Keywords
Set of Features Containing The Keywords
Explore The Filtered Records and Assign The Group to The Suitable Features
Set of Features Assigned to a Feature Group
Refine The Group and The Keywords
Refined Keywords for a Feature Group
Adapt The Filter in The Database
Set of Features Containing Group Keywords
Define Possible Subgroups and The Corresponding Keywords
Feature Subgroups
Assign The Subgroups as Groups to Sets of Suitable Features
Set of Features Assigned to a Feature Subgroup
Clusters of Features (Groups and Subgroups)
Set of Features Assigned to Hierarchical Groups
Assign The Group as Super-group to The Previous Set of Features

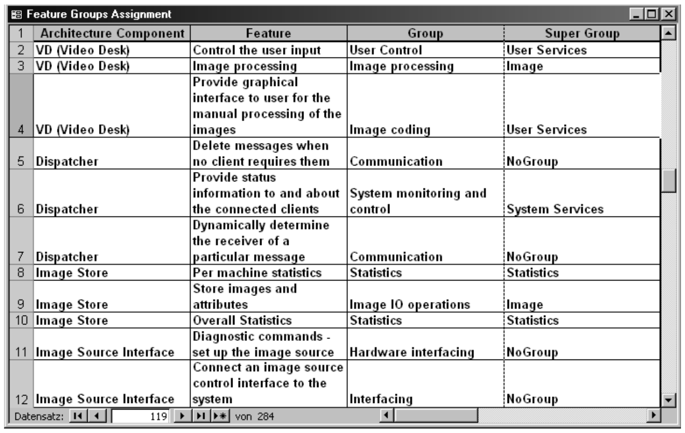
Figure 5. Features Clustering Process Data flow



	ID	Group Name	Keywords
	1	NoGroup	
▶	2	File IO Operations	Load, Save, Store, Delete, File, Container, Erase, Disk
	3	Image processing	Image, Cut, Rotate, Zoom, Region, Find, TIFF, Convert, Compress
	4	Communication	Interface, Message, Transfer, Receive, Send, Communicate, Protocol, Cor
	6	Statistics	Statistic, Counter, Report, Collect
	7	Result management	Results, Attributes
	8	Image IO operations	Image, Receipt, Retrieve, Store, Recover, Delete
	10	Image	Image, Attribute, tiff, source, request
	11	Diagnostics and testing	Monitor, Log, Diagnose, Trace, Debug, Test

Datensatz: 2 von 21

Figure 6. Features Clustering Tool - Feature Groups Edit Form



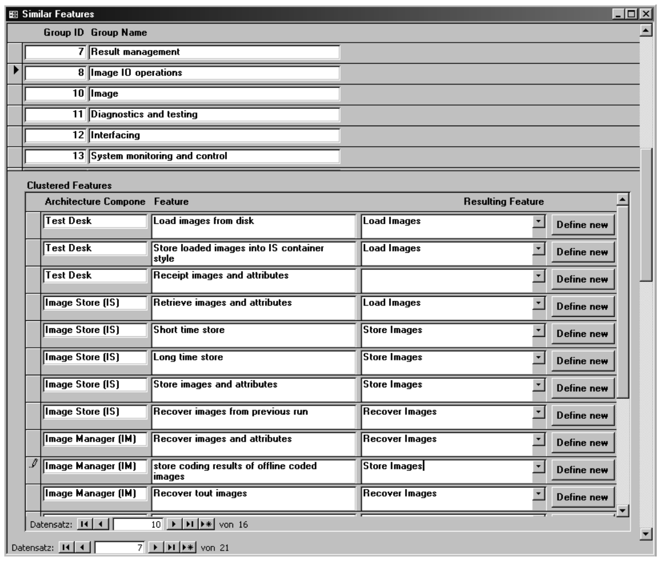
1	Architecture Component	Feature	Group	Super Group
2	VD (Video Desk)	Control the user input	User Control	User Services
3	VD (Video Desk)	Image processing	Image processing	Image
4	VD (Video Desk)	Provide graphical interface to user for the manual processing of the images	Image coding	User Services
5	Dispatcher	Delete messages when no client requires them	Communication	NoGroup
6	Dispatcher	Provide status information to and about the connected clients	System monitoring and control	System Services
7	Dispatcher	Dynamically determine the receiver of a particular message	Communication	NoGroup
8	Image Store	Per machine statistics	Statistics	Statistics
9	Image Store	Store images and attributes	Image IO operations	Image
10	Image Store	Overall Statistics	Statistics	Statistics
11	Image Source Interface	Diagnostic commands - set up the image source	Hardware interfacing	NoGroup
12	Image Source Interface	Connect an image source control interface to the system	Interfacing	NoGroup

Datensatz: 119 von 284

Figure 7. Features Clustering Tool - Feature Groups Assignment Form

Super Group	Group	Component	Feature
Image	Image IO operations	IS (Image Store)	Retrieve images and attributes
			Store images and attributes
			TD Store loaded images into IS container style
	Image TIFF operations	IDB	Store images in TIFF format
		II	Convert raw image data into standard format (TIFF)
		IR	Compress received images (TIFF FAX4)
		ISIF	Compress images (TIFF FAX4/JPEG)

Figure 8. Features Clustering Tool - Groups-Features Report (Established Feature Clusters)



	Group ID	Group Name
	7	Result management
▶	8	Image IO operations
	10	Image
	11	Diagnostics and testing
	12	Interfacing
	13	System monitoring and control

Clustered Features

Architecture Compone	Feature	Resulting Feature
Test Desk	Load images from disk	Load Images
Test Desk	Store loaded images into IS container style	Load Images
Test Desk	Receipt images and attributes	
Image Store (IS)	Retrieve images and attributes	Load Images
Image Store (IS)	Short time store	Store Images
Image Store (IS)	Long time store	Store Images
Image Store (IS)	Store images and attributes	Store Images
Image Store (IS)	Recover images from previous run	Recover Images
Image Manager (IM)	Recover images and attributes	Recover Images
Image Manager (IM)	store coding results of offline coded images	Store Images
Image Manager (IM)	Recover tout images	Recover Images

Datensatz: 10 von 16

Datensatz: 7 von 21

Figure 9. Features Clustering Tool - Similarities Examination Form