

User Directed Search Based Reverse Engineering

Frederik Schmidt

Student-ID: 787975

A thesis submitted to Auckland University of Technology in partial
fulfilment of the requirements for the degree of Master of Computer and
Information Sciences (MCIS)

2009

School of Computing and Mathematical Sciences

Primary Supervisor: Andy Connor

Secondary Supervisor: Stephen MacDonell

Abstract

The current research represents the planning, design, implementation and evaluation of a user directed software clustering approach that utilizes Search Based Software Engineering (SBSE). The aim of this research is to examine if a user directed software clustering approach contributes to the quality of software clustering. Because of the explorative and constructive character this research project utilises the System Development Research Methodology.

This research is enabled by the implementation of the Search Based Reverse Engineering (SBRE) component. The SBRE component features multiple similarity measurements and the inclusion of user constraints in the clustering process to create different implementation perspectives of the software system depending on the requirements and preferences of the stakeholders. These similarity measurements are based on software metrics, which measure different software-attributes. The SBRE component utilizes a greedy and tabu search algorithm for the identification of the cluster landscape of the analyzed software systems.

The evaluation showed that a user controlled SBSE cluster approach is able to adapt to different user configurations and derive corresponding cluster landscapes from software systems. Different measures are introduced to control the cluster process. It has been shown how these measures contribute to the quality of the clustering. It is demonstrated that tabu search is applicable in the field of software clustering. Finally, it has been examined that a multiple metric approach allows adapting the clustering process to the requirements of the stakeholders and the design of the software system to optimize the clustering result.

Attestation of Authorship

"I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning."

Yours sincerely,

Frederik Schmidt

Acknowledgements

There are many who supported to the completion of this work. To these individuals I express my most sincere thanks.

In particular, I wish to express my gratitude to my supervisors Dr. Andrew Connor and Prof. Steven MacDonnell for their continued encouragement and invaluable suggestions during this work.

I would also like to thank the staff and lecturers from the Auckland University of Technology, who supported me during the past two years of my studies. Furthermore, I would like to thank my postgraduate colleagues and friends for their personal encouragement and great support during this study.

Special thanks goes out to my partner for her patient and enormous support during good and tough times of this research. Her encouragement helped me through to the completion of this research.

Table of Contents

Abstract.....	II
Attestation of Authorship	III
Acknowledgements	IV
List of Figures	VIII
List of Tables.....	X
List of Algorithms.....	X
List of Abbreviations	XI
1 Introduction	1
1.1 Motivation and Background	1
1.2 Goal of the Research Project	2
1.3 Thesis Structure	4
2 Literature Review	6
2.1 Software Architectures.....	6
2.1.1 Artefact.....	7
2.1.2 Complexity	7
2.1.3 Cohesion and Coupling	9
2.1.4 Software Architecture Views	10
2.1.5 Ensuring Architectural Quality	12
2.2 Search Based Software Engineering.....	14
2.2.1 Classification of Metaheuristics	15
2.2.2 Implementation of Metaheuristics	16
2.2.3 Metaheuristic Algorithms	20
2.3 Software Clustering	23
2.3.1 The Development of Software Clustering.....	24
2.3.2 The Software Clustering Process.....	28

2.3.3	Inclusion of Expert Knowledge.....	34
2.3.4	Evaluation of Software Clustering Quality	35
2.4	Software Metrics	36
2.4.1	Classification of Software Metrics	38
2.4.2	Application of Software Metrics in SBSE.....	39
2.5	Summary.....	41
3	Research Objectives and Methodology	43
3.1	Research Objectives	43
3.2	Research Methodology	44
3.3	Research Design	45
4	Design and Implementation of the SBRE Component	50
4.1	Development Environment	51
4.1.1	The <i>Barrio</i> Framework	51
4.1.2	The <i>Bunch</i> Framework	52
4.2	Exposure of the Artefacts.....	53
4.3	Design and Implementation of the Similarity Measurements.....	55
4.4	Design and Implementation of the Cluster Algorithms	61
4.4.1	Solution Representation	61
4.4.2	Fitness Function	63
4.4.3	User Constraints.....	68
4.4.4	Metaheuristics	69
4.5	Functionality of the SBRE System.....	81
4.6	Summary.....	83
5	Evaluation of the SBRE component.....	84
5.1	Experiment Design	84
5.2	Evaluation of the Fitness Function	85

5.3	Evaluation of the Cluster Algorithms	90
5.3.1	Evaluation of the GreedyBestNeighbour Algorithm	90
5.3.2	Evaluation of the <i>TabuSearchStrategy</i> Algorithm	92
5.4	Evaluation of the SBRE Cluster Analysis	106
5.4.1	Evaluation of Multiple Implementation Perspectives	107
5.4.2	Evaluation of Rearchitecturing Functionality.....	114
5.4.3	Inclusion of User Domain Knowledge	117
5.4.4	Cluster Analysis on Different Abstraction Levels	119
5.5	Comparison of the SBRE component with <i>Bunch</i> and <i>Barrio</i>	121
5.6	Summary.....	124
6	Limitations, Future Research and Conclusion	125
6.1	Answer of the Research Questions	125
6.1.1	Application of the Tabu Search Algorithm	125
6.1.2	Multiple Implementation Perspectives.....	127
6.1.3	User Directed Software Clustering.....	129
6.2	Limitations	132
6.3	Future Directions.....	133
6.4	Conclusion	135
7	References	136
	Appendix A: CD – Enclosure	141

List of Figures

Figure 3.1 : Research process of the SDRM (Nunamaker & Chen, 1990)	45
Figure 4.1 : <i>Barrio</i> software clustering process	52
Figure 4.2 : UML class diagram of the SBRE ProjectSelectionTable	54
Figure 4.3 : Visualisation of the project selection component	54
Figure 4.4 : Illustration of the design of the SBRE metric framework	59
Figure 4.5 : Illustration of the metric configuration component	60
Figure 4.6 : Class diagram of the SBRE representation	62
Figure 4.7 : Illustration of the coupling between clusters	64
Figure 4.8 : Complexity of a system in relation to the granularity: adapted from Baas (2003)	65
Figure 4.9 : Class diagram of the SBRE cluster constraint framework	68
Figure 4.10 : Class diagram of the SBRE cluster framework	70
Figure 4.11 : Allocation into existing clusters of the SBRE greedy algorithm	72
Figure 4.12 : Assignment of artefact into a new cluster of the SBRE greedy algorithm	72
Figure 4.13 : Class diagram of the <i>TabuSearchStrategy</i> algorithm	75
Figure 4.14 : Example of the clustering process of the <i>TabuSearchStrategy</i> algorithm	77
Figure 4.15 : Example of the <i>TabuSearchStrategy</i> configuration component	80
Figure 4.16 : Screenshot of the SBRE component	83
Figure 5.1 : Initial solution of the fitness function experiment	87
Figure 5.2 : Solution of the fitness function experiment with a separated metric and cluster package	88
Figure 5.3 : Solution of the fitness function experiment with combined <i>cluster</i> and <i>junumberfield</i> package	89
Figure 5.4 : Runtime of the <i>GreedyBestNeighbour</i> algorithm depending on the artefact input size	92
Figure 5.5 : Configuration of the <i>TabuSearchStrategy</i> evaluation component	93
Figure 5.6 : Illustration of the metric configuration during the cluster algorithm evaluation	95
Figure 5.7 : Configuration of the maximal tested solutions experiment	96

Figure 5.8 : Solution quality in relation to maximum algorithm iterations of the SBRE system (package level)	97
Figure 5.9 : Solution quality in relation to maximum algorithm iterations of the SBRE system (class level).....	98
Figure 5.10 : Algorithm runtime in relation to number of artefacts	99
Figure 5.11 : <i>TabuSearchStrategy</i> configuration of the tabu list length experiment...	100
Figure 5.12 : Solution quality in relation to the length of the tabu list	101
Figure 5.13 : Rejected solution candidates in comparison to the length of the tabu list	101
Figure 5.14 : <i>TabuSearchStrategy</i> configuration for the diversification experiment...	103
Figure 5.15 : Solution quality in relation to idle diversification iterations (package level)	104
Figure 5.16 : Number of triggered diversification runs in relation to idle diversification iterations	104
Figure 5.17 : Solution quality in relation to idle diversification iterations (class level)	105
Figure 5.18 : Configuration of the <i>TabuSearchStrategy</i> algorithm within the multiple view experiment	107
Figure 5.19 : Result of the multiple view experiment with 100% weight on the CBO metric	108
Figure 5.20 : Result of the multiple view experiment with 100% weight on the CON metric	109
Figure 5.21 : Extract of the CON analysis cluster landscape of the „ <i>crm domain example</i> “	111
Figure 5.22 : Result of the deterioration cluster analysis with 100% weight on the static elements metric	113
Figure 5.23 : Metric configuration of the rearchitecturing analysis	115
Figure 5.24 : <i>TabuSearchStrategy</i> configuration of the rearchitecturing experiment .	115
Figure 5.25 : Solution of the rearchitecturing analysis of the SBRE component.....	116
Figure 5.26 : Solution of the SBRE component with cluster constraints	118
Figure 5.27 : Clustering of the SBRE system (class level)	120

List of Tables

Table 5.1 : Solution quality values of the fitness function experiment	89
Table 5.2 : Runtime in milliseconds of the <i>GreedyBestNeighbour</i> algorithm.....	91
Table 5.3 : Example of one result of the <i>TabuSearchStrategy</i> algorithm evaluation.....	94
Table 5.4 : Comparison clustering solution with 100% weight on the CBO or CON metric	110

List of Algorithms

Algorithm 4.1 : PseudoCode of the <i>GreedyBestNeighbour</i> cluster method.....	74
Algorithm 4.2 : PseudoCode of the <i>TabuSearchStrategy</i> spreadSolution method	78

List of Abbreviations

CDA	- Code Dependency Analyzer
CBO	- Cohesion Between Objects
CON	- Correlation of Names
GA	- Genetic Algorithms
GUI	- Graphical User Interface
IDE	- Integrated Development Environment
LoC	- Lines of Code
MDG	- Module Dependency Graph
MQ	- Module Quality
SA	- Simulated Annealing
SBRE	- Search Based Reverse Engineering
SBSE	- Search Based Software Engineering
SDRM	- System Development Research Methodology
SQ	- Solution Quality
TS	- Tabu Search

1 Introduction

This chapter presents the motivation for this research and derives from it the direction of the present research. A more detailed consideration of the research objective is given in chapter four after the research environment is further investigated with an in-depth literature review.

1.1 Motivation and Background

Forty years ago, the term ‘software crisis’ was first mentioned at the NATO Software Engineering Conference of 1968 (Randell, 1996). The term describes the circumstance of software systems being resistant to change and difficult to maintain. With unclear and changing requirements the result was a predominance of inflexible and unstable software systems (Glass, 2002). Many software projects were exhibiting problems due to these weaknesses.

The paradigms, processes, tools, computational platforms and techniques in the field of software engineering have changed immensely over the past 40 years, but the problems which have been summarized under the term ‘software crisis’ still exist. One reason for this is the high complexity of non-trivial software systems.

Contemporary software systems that comprise a reasonable (non-trivial) amount of functionality and size are invariably accompanied by a non-trivial degree of complexity (Bass, Clements, & Kazman, 2003). One reason for this complexity is the diversity of the artefacts (e.g. files, methods, classes, packages) involved in the software system. Furthermore, any given system structure is not continuous; the structure of the system changes through maintenance, requirements changes, added features and refactorings (Bosch, 2004). This creates difficulties for individuals attempting to understand the design, structures, dependencies and the architecture of a software system. As a result, realizing new requirements and maintaining a large software system is challenging. A good understanding of the system architecture is necessary to align the development of new requirements and the maintenance of the software system with the aspired system design (Bass et al., 2003).

If new functionality is added to an existing software system, without considering the software architecture or maintaining the integrity of the software system, then system erosion will occur. As a consequence software quality decreases and the system will be less flexible, less robust and harder to both maintain and to understand (Banker, Datar, Kemerer, & Zweig, 1993). Therefore the software maintenance cost increases. Finally, the system will be so substantially deteriorated that a complete rebuild will become necessary.

This general scenario illustrates the importance for development stakeholders to access current and consistent documentation of the architecture and structure of the software system in order to form a good understanding of the current system organization in a timely manner. Because of the evolving character of software systems and their often substantial scale it is certainly beneficial if this documentation could be created automatically.

An approach to increase the maintainability and understandability of software systems is to hide the complexity of a software system by abstraction. One implementation of this approach is called software clustering and arose in the field of reverse engineering (Chikofsky & Cross, 1990). Software clustering attempts to partition a software system into subsystems and thereby create a new level of system abstraction. Through these subsystems stakeholders could be provided with abstract information about design, structure, organization and dependencies of the software system.

1.2 Goal of the Research Project

The objective of the current research is to design, implement and evaluate an approach, which provides useful information to developers and architects about software architecture clustering, subsystem decomposition and problematic source code segments. This should enable the stakeholders, depending on their requirements, to derive a flexible and maintainable implementation structure of the system. Isolating *code smells* (Fowler, 1999) within software system could also assist the stakeholders to plan appropriate refactorings to eliminate longer terms problems arising.

To control and observe the development process and manage the complexity of non trivial software systems the application of software development tools is necessary. An emerging approach is to integrate software development components directly into the Integrated Development Environment (IDE). The advantage is that the necessary information is directly available when a task is planned or implemented. Contemporary IDEs offer extension possibilities to integrate external components into the development process to allow a flexible adaption to the stakeholder requirements. One framework for these extensions is the *Eclipse*¹ plug-in concept. This work follows these ideas and integrates the process of software clustering into the *Eclipse* development environment and obtains information for stakeholders regarding design, structure and dependencies analyses. Regarding the prototype characteristic of this project and the established popularity of object oriented software systems, this work focuses on the analyses of such systems and especially of *java* software systems. Considering the intent to support an integrated approach, one of the challenges of this project is to provide architecture information in a reasonable time. Regarding the size and increasing complexity of current software systems any solution should also feature effective scalability.

One field of research, which may support adherence to these requirements, is that of Search Based Software Engineering (SBSE), introduced by Harman and Jones (2001). SBSE describes the application of metaheuristic algorithms in the area of software engineering. Encouraging results have been achieved to date in the areas of testing, standalone cluster analysis, release-planning and requirement analysis.

The present research examines the application of Search Based Software Engineering (SBSE) in the area of software clustering with a special focus on user directed software clustering. The research of Harman and Jones (2001), Mitchell (2002) and Seng, Bauer, Biehl & Pache (2005) provides a degree of evidence that the computational challenges in the area of software clustering and subsystem decomposition can be solved with the application of SBSE. These studies lend support to the belief that an SBSE-based

¹ <http://eclipse.org/>

solution should provide the developer with architecture information after a sufficiently short processing time.

The modularization of software systems as a basis for refactorings or rearchitecturing can utilize a variety of variables. Because of the multi-objective nature of software development these objectives can often be in direct conflict with each other. Considering this, it can be said that no universal solution can be created to solve a given problem. Because of a focus on different attributes or even personal preferences there can be a range of solutions that are all “equally optimal”. To create a flexible solution, which adapts well to the reviewed system and the preferences of the stakeholders the inclusion of specific user knowledge to control the cluster process is necessary. Hence a clustering approach should not aim for a final and perfect solution. It should rather aim for a flexible solution that can be quickly created and easily manipulated.

Based on an in-depth literature review in chapter three and the identification of challenges and limitations of the relevant studies the following research questions are examined during the present research:

- Can a user directed and semi-automatic clustering approach contribute to the quality of software clustering?
- Is tabu search applicable in the area of software clustering?
- Does the inclusion of multiple metrics in the fitness function enable the clustering of a software system into multiple implementation perspectives?

Based on this motivation and broad illustration of the direction of the research the following section provides an overview of the structure of this thesis.

1.3 Thesis Structure

This work is subdivided into six chapters. Chapter one illustrates the motivation and placement of the research. The remaining chapters of the thesis are structured as follows: Chapter two examines relevant research in the areas of this research and introduces relevant terms which are necessary for the understanding and development of this work. Based on this literature review, the identification of

limitations of the current research and illustration of a promising research path, the final research objective and research questions are depicted in chapter three. Additionally, the research methodology and design of the research is illustrated within this chapter. Chapter four illustrates the design and implementation of the research objective, which is utilized by the Search Based Reverse Engineering (SBRE) component. Chapter five demonstrates the evaluation process and portrays the results of this evaluation and discusses and aligns them with the research questions. Finally, Chapter seven presents the conclusions for this research, highlights the contribution of this study to the related research fields, considers the limitations of this study and provides recommendations for future research.

2 Literature Review

This research project draws on a number of different fields within software engineering particularly software clustering and search based software engineering. Furthermore, the concepts and terms used in the field of software architecture are important in supporting an understanding of the purpose of this research. This literature review considers these areas with special attention to the proposed research. The area of software architecture can be seen as the application domain of this work and hence is addressed first.

2.1 Software Architectures

Software architectures provide a framework for the development of software systems (Bass et al., 2003). The architecture is an orientation within which the developer is able to create a uniform software design. The program design should remain consistent so developers who are familiar with the architecture can understand the design of other system parts. In theory, effective software architectures support the continuous conservation of maintainability and extensibility that can prevent the erosion of the software system. But certainly not all software architectures are similar. Different factors influence the design of a software architecture e.g. performance, modularisation, system distribution. Not all of these factors correspond with the requirement to preserve maintainability and extensibility.

A technique that enables developers to obtain insight into non-trivial software systems is the decomposition into modules (Courses & Surveys, 2002). This modularisation can be driven by different aspects. Possible classifications can be to follow a functional, object oriented or data driven decomposition (Bass et al., 2003). A software architecture portrays the structure and the decomposition of a software system into components and the relations between these components. To define a software architecture a meaningful decomposition of subsystems, according to the domain-, system- and quality- requirements, has to be identified. Additionally, the dependencies between these subsystems have to be defined. Effective software architectures should help developers to control the stability of large systems, which feature a high degree of complexity (Bass et al., 2003). This is enabled through the

modularisation, classification and aggregation of smaller unstructured elements into bigger controlled structures.

In the next sections relevant terms and definitions used in the area of software architectures are introduced.

2.1.1 Artefact

One goal of software architecture development is the abstraction of software systems. Software systems feature different levels of abstraction (e.g. files, folders, methods, classes, packages). For some design rules and analyses the application level is irrelevant. For example, the maxim of low coupling is relevant on class and package level. To express these generalisations the term artefact is introduced. The term artefact allows discussing problems and solution strategies on an abstract level. Within this work the term artefact abstracts the design levels class, package and subsystem. A subsystem is an aggregation of artefacts, which exhibits a certain similarity.

2.1.2 Complexity

The term complexity in the context of the present research relates to the level of difficulty encountered by a developer, who is attempting to understand a software system (Zuse, 1991). A high level of complexity causes problems for the maintenance and modification of software systems. This aspect is especially significant because maintenance cost is one of the main cost factors in software projects (Banker et al., 1993). Darcy and Kemerer (2002) listed possible reasons for high complexity in a software system: complexity and difficulty of the problem itself, an inappropriate solution design, a high coupling between artefacts or even a high flexibility of the software system itself.

Complexity Classifications

In the software engineering literature different classifications of software complexity can be found. To outline the field of software complexity this work utilises the classification introduced by Fenton and Pfleeger (1997), which is taken up in other established research papers, e.g. Van Vliet (2000) and Mens and Demeyer (2001) .

Fenton and Pfleeger (1997) classified software complexity into four categories: *algorithmic*, *structural*, *cognitive* and *problem* complexity. Structural complexity depicts the complexity of relations and dependencies between artefacts and the consideration of these structures in the context of the complete software system. These relations are of special importance for cohesion and complexity analysis. The *algorithmic complexity* portrays the intricacy of a particular algorithm. The *cognitive complexity* describes the effort for a person to understand an artefact of a software system. The estimation of cognitive complexity underlies a quite subjective and individual assumption. Cognitive complexity can be considered as the inverse to the understandability software quality attribute. The subjective and individual factors of cognitive complexity make the investigation and estimation of cognitive complexity difficult. The *problem complexity* expresses the complexity of an optimal algorithm to solve a problem. The problem complexity is always smaller or equal to the algorithmic complexity.

Computational Complexity

Besides the classification of Fenton and Pfleeger (1997) the term *computational complexity* exists. Because of the focus of this work computational complexity is of special importance and will be further illustrated in this section.

The term computational complexity should not be mixed up with the algorithmic complexity. As mentioned previously, algorithmic complexity describes the difficulty of an algorithm and the difficulty of understanding a certain algorithm.

The computational complexity defines the needed time and space resources to solve a problem depending on a certain input size (Zuse, 1991). It is the aim of the computational complexity analysis to estimate the resource demand regarding the change of the input size. It is not the goal of the computational complexity analysis to give an exact step count of a certain algorithm, but rather give an asymptotic step count of the algorithm. The time complexity defines the number of required steps to solve a problem of the input size n (Yang, 2008). The time complexity is usually given for the worst case scenario.

Referring to Yang (2008) an algorithm is tractable, if it is solvable in polynomial time. On the other hand it is intractable, if no algorithm exists which can solve the problem in polynomial time. A problem is solvable in polynomial time, if an algorithm exists which is solvable on a deterministic and sequential computer and if the calculation time does not grow stronger with the input problem size than a polynomial function. Consequently, the term polynomial time distinguishes whether a problem is practically solvable or not. Even with a relatively small input size the computational complexity of an intractable problem will grow so quickly that a practical application of this algorithm is impossible.

Importance of Complexity

It has to be considered that the different complexities influence each other. A very complex algorithm has also a negative effect on the cognitive and structural complexity. Hence it is not possible to examine the certain complexities individually.

The computational complexity of the cluster partitioning problem is of particular relevance to this research as it applies Search Based Software Engineering (SBSE) in the field of cluster partitioning. The field of SBSE is depicted in section 2.2. The field of cluster partitioning and cluster analysis is further described in section 2.3. However, the aim of this work is to assess and reduce predominantly structural complexity by using clustering and abstraction techniques in a way that does not exceed polynomial bounds of computational complexity. As a consequence of the reduction of structural complexity the cognitive complexity imposed on developers should also be reduced by these refactorings.

2.1.3 Cohesion and Coupling

The terms cohesion and coupling are of central importance for the decomposition and modularization of software systems (Darcy & Kemerer, 2002). The coupling measurement defines the strength of binding between two artefacts. A higher number of dependencies and interconnections between two artefacts increases the coupling between these artefacts. A high coupling between two artefacts hinders the reusability and understandability of these artefacts (Gui & Scott, 2006).

Cohesion describes the internal coherence of an artefact (Counsell, Swift, & Crampton, 2006). A reason for a low level of cohesion could be the implementation of two different functionalities within an artefact. The semantics of the functionality provided in the artefact is therefore not coherent. This inclusion of multiple functionalities hinders reusability and understandability of the affected artefact.

In relation to these characteristics the aims of modularization are low coupling between artefacts and high cohesion within each artefact.

2.1.4 Software Architecture Views

The main aim of architectures is the reduction of complexity (Taylor & Van Der Hoek, 2007). All the information relating to any non-trivial architecture could not generally be illustrated in one view, as a reduction of complexity would certainly not be possible. To investigate certain attributes of software architectures different views of a software system exist. A view depicts a software system from a certain architectural view. If details are not relevant for a certain view, they are abstracted or excluded. In the literature there is no general classification of architecture views. Bass et al. (2003) suggested a division into the five architecture views: conception, infrastructure, implementation, runtime and data architecture views. Within the present research the implementation view is of special importance. However, the other views are also briefly introduced to allow the differentiation of the individual views.

The *conceptual view* displays the functionality of the system in an abstract manner. The system is portrayed in its logical system environment. System borders, user and surrounding systems are visualized in interaction with the system. The internal system workflows are not displayed. The *infrastructure view* documents the platform and system environment of a system. These could be computers, processors, network topologies or other parts of the physical system environment. The *implementation view* illustrates the internal configuration of a system on the level of software artefacts. The system is distinguished into components to a particular level of detail. Depending on the programming language and paradigm, artefacts within the implementation view can be subsystems, components, packages, classes or files. Further on, the interfaces and dependencies between the individual artefacts are

illustrated within the *implementation view*. The *runtime view* describes which instances of artefact exist during the execution of a component and the interaction of these components between each other. Finally, the *data view* illustrates the structure of data elements and their interrelationships.

Different views are appropriate or relevant to analyse a system regarding a certain aspect or requirement. It also has to be illustrated that these views do not ensure the architectural quality of a software system. They rather display the actual condition of the software system from a certain perspective. Thus an architectural view does certainly help to understand and analyse a software system, but does not conserve the architecture of the system. As depicted in the beginning of the section, one aim of software architectures is to anticipate the deterioration of a software system. Certainly the possibility to analyse and understand a system contributes to a design which aligns with the actual architecture, but it does not ensure that this complies with the architecture definition. The architecture definition describes the decomposition and dependencies of a software system (Shaw & Garlan, 1996). The architecture definition is therefore a description of the target architecture. The design of current and future implementations should align with this target architecture. The compliance with the target architecture should obtain a robust, clearly arranged, modularized, maintainable and expandable software system (Bass et al., 2003).

It is certainly possible to define a target-architecture for every architecture view of a software system. However, as described in section 1.1 the main problem of current software systems are the high maintenance costs caused by the deterioration of the software system. This illustrates that the definition and observation of the implementation view is of special interest within the present research.

Further on, the process of architecture observation to ensure the architectural quality is illustrated and discussed to highlight the actual problems with these approaches.

2.1.5 Ensuring Architectural Quality

Different workflows exist to monitor and ensure the quality of a software-architecture (Bass et al., 2003). Often, approaches to monitor and obtain the software architecture deal with the implementation of tasks and the observation of the software architecture separately (Bischofberger, Kuhl, & Loffler, 2004; Hofmeister, Nord, & Soni, 2000; Salger, Bennicke, Engels, & Lewerentz, 2008). These approaches suggest that a developer is focusing on the completion of tasks. The Integrated Development Environment (IDE) supports the developer to finish these tasks. The IDE usually provides limited information regarding the desired software architecture or to support the creation of a design with high quality. Often a software architect later assesses the committed changes and identifies possible architecture violations and code smells (Bosch, 2004). An architecture violation arises when resources are accessed in an inconsistent manner to the architecture definition. The term *code smell* (Fowler, 1999) describes the occurrence of symptoms within the source code that could indicate a deeper design problem. The elimination and prevention of code smells and architecture violations should abolish, or at least reduce, the deterioration of the software system and obtain the above mentioned robust, clearly arranged, modularized, maintainable and expandable software system.

Bischofberger, Kuhl and Loffler (2004) introduced approaches to identify architecture violations and to obtain a preferred architecture of a software system. They are based primarily on the comparison of an architecture definition with the actual system source code. The comparison between the target architecture and the current architecture identifies architecture violations.

The identification of code smells relies extensively on software metrics, which indicate possible problematic code segments and artefacts. Based on these indications a visual code inspection should identify the problem and a refactoring can be planned.

The architect creates tasks for the detected architecture violations and code smells and the developer has to implement these tasks. This procedure uncouples the quality assurance process and the implementation process. Unfortunately, this approach slows down the development process and creates turnaround times. This motivates

that an early identification or prevention of architecture violations and code smells increases the productivity and decreases the costs of software system development. A similar link between the time of software failure identification and the effect on development costs is well known knowledge in the field of software engineering and drawn in various publications (Knox, 1993; Pham, 2003; Swanson, 1976).

This overhead could be decreased, if the developer has a good understanding of the software system and information that would help him to prevent architecture violations and code smells. The architecture monitoring process should be integrated directly into the IDE to decrease turnaround times. The advantage should be that the developer is getting information about the software architecture and the influences of his changes instantly and as consequence produces fewer architecture violations and *code smells*.

The consideration of architecture monitoring raises the question of how to define the target architecture. Certainly the definition of a target-architecture within newly developed systems with a well planned design would normally be relatively easy. But within older systems, which have grown and changed over a long period of time, the definition of a target architecture can be much harder.

This current work attempts to address this challenge. The objective of this research is to derive a structure of the current system architecture and identify clusters within that system. The intent is to provide the developer with a sufficient understanding of the system organization in a timely manner. This would enable them to identify code smells and produce fewer architecture violations. Identification of the dependency relationships among components should help the developer to understand the connections and identify subsystems of the software system. A subsequent step would be to classify the extracted artefacts in a system specific architecture raster. Only this alignment of the actual architecture and the desired architecture gives the possibility to identify a lack of quality e.g cycles, or architecture violations, and to derive quality improvement opportunities (Bass et al., 2003). However, the application of this component will not remove the necessity of an appropriate architecture definition and monitoring.

2.2 Search Based Software Engineering

This research focuses on the application of Search Based Software Engineering (SBSE) in the area of software clustering. Accordingly, this section illustrates the fundamental characteristics and concepts of SBSE as well as exposing some of the most pertinent literature in this field.

For some problems in the domain of software engineering e.g. release planning, clustering and testing more or less optimal solutions exist. From a formal viewpoint it is possible to identify the optimal solution to a problem in each of these contexts. However, the computational complexity to achieve this optimal solution may be very high if the solution space (i.e. the number of potential solutions) is large. Instead a good or nearly optimal solution that can be found more quickly than the optimal solution may be sufficient. Harman and Jones (2001) introduced SBSE as the application of metaheuristic algorithms to solve linear optimization problems in the area of software engineering.

SBSE has emerged as a vibrant research topic with evidence in the literature showing that it is widely applicable across the whole spectrum of lifecycles activities e.g. from requirements engineering (Bagnall, Rayward-Smith, & Whittle, 2001), project planning and estimation (Burgess & Lefley, 2001), refactoring and maintenance (Harman & Tratt, 2007; O’Keeffe & Ó Cinnéide, 2008), testing (Ribeiro, Rela, & Vega, 2008; Wegener, Baresel, & Sthamer, 2001) and quality assurance (Khoshgoftaar, Yi, & Seliya, 2004).

A key element of an SBSE implementation is the choice of search algorithm used. A wide range of search techniques could be applied, including gradient methods, direct search or metaheuristics. Metaheuristics have become more common in recent literature, because the complexity of many problems does not suit gradient or direct search methods.

The term *heuristic* refers to problem solving strategies, which apply general sense and assumptions and loosely applicable information to arrive at a nearly optimal solution in a relatively short period of time (Yang, 2008). Heuristic algorithms are a branch of

operations research. Heuristics are often applied, if there is no formal method known to calculate the optimal solution or there is a formal way known but the computational complexity exceeds polynomial time.

Heuristic algorithms can be distinguished into specific heuristic algorithms and metaheuristic algorithms. Lin & Kernighan (1973) introduced for example a specific heuristic algorithm to solve the travelling salesman problem. In comparison to specific heuristic algorithms metaheuristic algorithms are designed more generically to be able to solve different problems (Yang, 2008). The term 'meta' relates to 'upper' or 'higher level methodology'.

Talbi (2009) suggested a range of classifications of metaheuristics in terms of nature inspired vs. non-nature inspired, memory usage vs. memory less methods, deterministic vs. stochastic and iterative vs. greedy metaheuristics. To position this research and provide sufficient detail regarding the diversity of algorithms this classification will be constituted.

2.2.1 Classification of Metaheuristics

Many metaheuristic algorithms are inspired by the behaviour of animals or natural processes. The most important of these nature inspired algorithms are briefly described here to give an idea about their inspiration. *Evolutionary Algorithms* utilize the model of biological evolution and copy the three main biological principles mutation, recombination and selection (Bäck, 1996). The best adapted individuals form the solution representation in keeping with the principle of survival of the fittest. *Evolutionary Algorithms* are distinguished into the four categories of genetic algorithms, evolution strategies, genetic programming and evolutionary programming (Bäck, Fogel, & Michalewicz, 1997). *Ant Colony Optimization* clones a strategy which can be found in the complex social behaviour of ants during their forage (Dorigo & Stützle, 2004), whilst *Particle Swarm Optimization* problems based on the forage process of bird- and fish swarms (Kennedy, Eberhart, & Shi, 2001). *Simulated Annealing* is based on an analogy from thermal substance physics, namely the crystallization in metals, which are released out of the melting process and are in a controlled cooling down process (Kirkpatrick, Gelatt, & Vecchi, 1983, p: 670).

Some metaheuristic algorithms are stateless and store no information of previous iterations or solutions to guide their next search step or search in general. One example of a metaheuristic which does not use any memory is the greedy algorithm. In contrast, an example of a metaheuristic which does use memory is the tabu search, which employs both short- and long-term memory as tabu lists (Glover, 1989).

Another classification mechanism relies on the distinction between deterministic and stochastic algorithms. Based on the same search space and the same search parameters a deterministic metaheuristic will find the same solution in multiple runs. Examples for deterministic metaheuristics are hillclimbing and tabu search. In stochastic algorithms the application of random rules could lead to different solutions even within the same starting configuration. Examples of stochastic algorithms are simulated annealing and evolutionary algorithms.

Finally, the last classification is based on the distinction between iterative and non-iterative algorithms. Iterative metaheuristics start with an initial solution. Based on this initial solution the algorithm tries to improve the solution. Examples of iterative algorithms are hill climbing and tabu search. In comparison the greedy algorithms start their search with an empty solution (DeVore & Temlyakov, 1996).

2.2.2 Implementation of Metaheuristics

Certain components and steps have to be considered when developing and applying metaheuristics for a given problem. These components relate to the representation of the solution, the determination of the fitness function, the handling of constraints, the interaction of users and finally a process to implement the metaheuristics.

Representation

Iterative metaheuristics manipulate solutions to derive new solutions. It is therefore necessary that these algorithms incorporate a suitable representation for these solutions (Glover & Kochenberger, 2003). An instance of the representation reflects a specific solution within the solution space. The term solution space describes all possible solutions of a certain optimization problem. The representation has to be aligned with the optimization problem and the applied operators, so that it is indeed

possible to manipulate the solutions. An operator is a function that transforms the current candidate solution from the solution space into another solution in the same solution space. The representation is an important component for the efficient and effective application of metaheuristics (Birattari, 2005).

A useful representation fulfils the requirements of completeness, connectivity and efficiency (Birattari, 2005; Glover & Kochenberger, 2003). Completeness describes the necessity of the representation to express every solution of the solution space. Connectivity describes the requirement of a search path between every two solution representations. That is, that for every solution $S1$ of the search space exist at least one set of operations for any other solution of the search space $S2$, which, applied in a certain order, transforms $S1$ into $S2$. Finally efficiency describes the requirement that the handling of the representation should be minimal in reference to the time and space complexity.

The Fitness Function

Whilst metaheuristics can produce a variety of different solution candidates, out of these the “best” solution has to be identified. This raises the requirement for a method to compare solutions with each other. This requirement is satisfied by the fitness function (also referred to as the objective function) (Harman & Jones, 2001).

The fitness function f quantifies the optimality of a solution (Harman & Jones, 2001) and therefore expresses the goal of the search. The fitness function is a surjective transformation, which maps the solution space into the set of real numbers $f: S \rightarrow \mathbb{R}$. The fitness function allows a ranking of solutions with each other. A fitness function consists of a combination of different factors, which reflect the quality and value of a given solution. As such, the quality of the fitness function depends on a sound orchestration and estimation of the influencing factors. If the fitness function is not well designed it will lead to unacceptable or poor solutions.

Constraint Handling

With reference to Talbi (2009) many optimization problems are constrained. Talbi (2009) introduces five strategies to handle constraints within metaheuristics, namely

reject, penalizing, repairing, preserving and decoding. These constraint strategies work either on the objective function, the representation of solutions or are included into the metaheuristic algorithm itself. The penalizing and preserving strategies are of particular importance for the application of metaheuristics in the area of software clustering and so are further discussed in this section.

Depending on by how much a constraint is violated, some solutions in a given solution space are more feasible than others. Often patterns can help to identify infeasible solutions (Talbi, 2009), and these infeasible solutions can be penalized. One way to implement these penalties is to add a linear factor to the fitness function f and obtain a new fitness function f' , which includes the penalizing strategy.

$$f'(s) = f(s) + \delta(s)$$

Preserving constraints are another important constraint strategy for the application of metaheuristics in the area of software-clustering. Preserving constraints obtain the feasibility of a solution considering a set of constraints (Glover & Kochenberger, 2003). The aim is that every preserving constraint is valid for each generated solution. As mentioned above, the preserving constraints have to be considered during the creation of the solution, which implies that the handling of preserving constraints has to be included in the metaheuristic algorithm. An example of a preserving constraint in the area of software clustering could relate to the requirement for a 'locked' cluster, comprising certain entities. The configuration of this cluster should be equal for every generated solution, if the lock constraint is set for this cluster.

Interactive Optimization

For some optimization problems the addition of certain information to improve the solution quality *during the search* can be beneficial. Birattari (2005) described the possible interactions in terms of two categories: User interaction to guide the search process and user intervention to evaluate a solution. The user can influence the search process to guide the search process into more promising areas. The aim is to improve the efficiency and performance of the search and to hopefully improve the solution quality. User intervention to evaluate a solution does not influence the search process.

Rather it leverages user information where no formal definition exists to quantify a solution. An example would be to evaluate the visual appeal, taste or attractiveness of an attribute or solution.

Relating to the application of metaheuristics in the area of software clustering it would be conceivable to allow the inclusion of special design knowledge to guide the search. The design of a software system occurs under a variety of external influences and personal preferences and as such informed by a wide variety of tacit knowledge that is difficult to encapsulate in mathematical form. Thus to recover the complete design of an existing system in an entirely automatic manner is a challenging problem. Given this, the user should have the chance to include domain and design knowledge into the search process.

The Design Process of Metaheuristics

In Talbi (2009) a set of guidelines is given for the application of metaheuristics. The first step is to model the problem. Based on this model and the nature of the problem an assessment of whether metaheuristics are applicable to this problem can be made. Factors that influence this decision are the computational complexity, size and structure of the problem and also the requirements of the application. Important requirements for a metaheuristic-based approach may relate to search time, quality of solutions, and solution robustness. When a metaheuristic-based approach is considered applicable the metaheuristic can be designed. Referring again to Talbi (2009), the representation of the problem, guiding of the objective function, constraint handling and selection of the algorithm need to be defined within the design phase. Based on the design, the metaheuristic-based approach can then be implemented. After the implementation the performance evaluation and parameter tuning phases are conducted. These phases are interconnected and can influence each other. This evaluation may reveal drawbacks with the search or the solution(s), which may make it necessary to change the model, the design and/or the implementation. Reflecting on, Talbi (2009) suggested an iterative and incremental approach for the development of metaheuristic-based applications.

2.2.3 Metaheuristic Algorithms

A variety of different metaheuristic algorithms exist. The various algorithms all aim to achieve the same end result in different ways. The intent is to carry out an efficient, yet effective search of the solution space in order to identify the best (and hopefully close to optimal) solution. This intent is embedded in the concepts of diversification and intensification. In the design of a metaheuristic the two conflictive parameters of exploration of the search space (*diversification*) and exploitation of the best solutions (*intensification*) have to be balanced (Glover & Kochenberger, 2003). The term *diversification* describes the necessity to explore non-visited areas to ensure that all regions of the search space are explored and that the search is not limited to only a small number of regions. *Intensification* describes the more focused exploration of promising areas that have delivered already good solutions. Certainly some metaheuristic algorithms focus more on diversification and others more on intensification. An example of an algorithm that focuses on diversification is the random search (Yang, 2008). The random search algorithm is creating a new random solution within every algorithm-iteration without considering previous solutions. In comparison to the random search the basic local search (hillclimbing) algorithm selects the best neighbour solution and is an example of a strong intensification approach.

The number of defined metaheuristics is large and the selection of a suitable metaheuristic depends on a variety of different requirements e.g. quality of the solution, possibilities for the manipulation of the search process and the acceptable complexity of the search (Birattari, 2005).

The greedy- and tabu search (TS) metaheuristics have been chosen for the implementation within this research project. The reasons for this are a lack of application in the area of software clustering within previous research (Jiang, Gold, Harman, & Li, 2007; Mitchell, 2002; Seng, Bauer, Biehl, & Pache, 2005), a high flexibility and controllability of the tabu search algorithms (Glover & Kochenberger, 2003) and a good performance of the greedy algorithms (Yang, 2008). A more detailed illustration of the motivation, reasons and relevance for the application of these algorithms within

this research is given in section 2.3. The *tabu search* and *greedy* metaheuristics demand special attention and will be illustrated in the next sections.

Greedy Algorithms

Greedy algorithms follow the problem solving metaheuristic of considering only the next possible steps at each stage within the search and selecting the best step of this set of possibilities (Cormen, Leiserson, Rivest, & Stein, 2001). The greedy algorithm chooses the step which promises the highest reward. By applying this approach the greedy algorithms tries to find the global optimum.

The advantages of greedy algorithms over alternatives are that they are easy to design, command low complexity and deliver solutions in a relatively short period of time (Glover & Kochenberger, 2003). The disadvantages of greedy algorithms are that the quality of the delivered solution can be very poor and they provide no backtracking to reverse decisions (Glover & Kochenberger, 2003). Additionally, greedy algorithms have a tendency to stall in local optima. A local optima characterizes a current solution, which cannot be improved by the applied metaheuristic (Glover & Kochenberger, 2003). The metaheuristic is not able to manipulate the solution candidate to escape the local optima and find the globally optimum solution. But this solution (local minima) is not the global optimum. To overcome this problem multiple start constellations can be used. This approach creates a solution for every start point. After the creation of this set of solutions the best solution is selected (DeVore & Temlyakov, 1996). Referring to Voßs, Fink and Duin (2005) special greedy algorithms (e.g. pilot method) incorporate look-ahead features to evaluate the consequences of a planned decision.

Iterative metaheuristic algorithms assume a start solution, which is manipulated by the algorithm. This raises the question how this start solution is generated. The greedy algorithm starts with an empty solution (DeVore & Temlyakov, 1996). Based on this, the algorithm adds every step the most appropriate decision until a final solution is found.

Tabu Search Algorithms

Tabu Search is a metaheuristic algorithm introduced by Glover (1989) and further developed in Glover & Laguna (1997). Glover (1990) illustrated a deterministic algorithm, which escapes local optima inspired by the controlled randomization applied in *Simulated Annealing (SA)*. The particular feature of tabu search is the use of memory which stores information related to the search process. Tabu search improves the navigation during the local search to find better solutions. This is achieved by forbidding solutions, which are not promising or which have been investigated before. Tabu search replaces the current best solution, as hillclimbing, if a neighbour features a better solution quality. If a local optimum is reached and no improving neighbour can be selected then tabu search selects a non improving neighbour to continue the search. This approach itself creates cycles and the algorithm would not terminate properly. To overcome these cycles the visited solutions are stored in the short-term tabu list. Solutions which are stored in the short term-tabu list are not visited again by the algorithm, as evolution potential of this branch is already investigated. The tabu list is updated with every selection of a new current solution. The algorithm has to check every algorithm-iteration, if a solution candidate is already in the tabu list. These continuous checks of all previously visited solutions are time and space consuming. This challenge desires the implementation of a small and easy to identify solution representation. Another method to reduce the computational complexity of the algorithm is to limit the maximum size of the tabu list. This approach prevents cycles with a maximum length of the maximum number of elements within the tabu list. Another way to reduce the computational complexity of tabu search is to store a record of the applied moves (Glover & Kochenberger, 2003). These moves are stored in relation to the created solutions. A move m which creates a new solution s_i based on a solution s_j is not allowed for a number of iterations. The inverse move m^{-1} , which creates the solution s_j based on the solution s_i is also blocked for a given number of iterations. Referring to Glover (1995) the number of iterations for which a certain move is blocked is called tabu tenure.

Another concept of tabu search is the application of aspiration criteria (Glover & Kochenberger, 2003). These criteria allow the selection of a solution, even when this

solution is within the tabu list. The reason for the application of aspiration criteria is that some tabu search implementations restrict attributes of solutions. This can exclude a large area of the search space. An easy aspiration criterion is for example the acceptance of all improving solutions, even if they are tabu.

Glover (1989) also illustrated the concept of medium-term and long-term memory illustrated to balance the conflictive parameters intensification and diversification. The medium term memory records the outstanding solutions. Similarities within these most promising solutions are identified. Based on these similarities the search can be intensified in the area of solutions with these characteristics. The mechanism illustrated by Glover (1989) to encourage the diversification of the tabu search is the long-term memory. The algorithm records information about visited solutions along the search. This information could be for example: How often a certain component is included into the solution. This information can be used to guide the search into previously unvisited areas of the search space. For example: The algorithm could be forced to include components into the solutions which have not been included in any previous solution.

Tabu search is an especially interesting approach, because it combines and balances the conflictive parameter of intensification and diversification. Further on, it does overcome the problem of greedy algorithms and hillclimbing to stall in local minima and has been shown that a given implementation can have robust performance across a range of problems with different problem representations (Connor, Clarkson, Shahpar, & Leonard, 2000).

2.3 Software Clustering

This section illustrates the field of software clustering. Firstly the term and classification are illustrated. Secondly the development of the field software clustering and relevant research in this area is described. Additionally, the necessary steps for the development of software clustering approaches are demonstrated and a framework for the development of software clustering is introduced. Further on, in regard to the research objective the inclusion of user knowledge into the cluster process and

possible approaches for the evaluation of the clustering quality are demonstrated. Finally, the relevant research studies are analyzed and related to this research project.

Software clustering is in essence a sub domain of reverse engineering (Chikofsky & Cross, 1990). The term software clustering describes the classification of a software system into partitions (Tzerpos & Holt, 2000). The artefacts of the software systems are distributed into different partitions according to measures of artefact similarity. These partitions are called clusters. These clusters are created by the identification of similarities between the artefacts. The clustering identifies similar artefacts and abstracts them into clusters with consideration of certain similarity attributes. Regarding this, the result of the clustering is a new abstraction level of the software system. A concrete representation of this clustering is called a cluster landscape. According to the application of metaheuristics in the field of software clustering during this research a cluster landscape represents a candidate solution. Additionally, another term which is of relevance for this work and will be used within this chapter is the term implementation perspective. The term implementation perspective should not be mixed up with the term implementation view (discussed in section 2.1.4). The term implementation perspective defines, within the present research, the clustering of the artefacts of the implementation view by certain criteria. These criteria define a specific perspective on the implementation view.

2.3.1 The Development of Software Clustering

The published literature in the area of software clustering shows that this has been an intense field of research during the last 15 years. Referring to Mitchell (2002) and Bass et al. (2003) the principal reasons for this are the migration from two-tiered to three-tiered and n-tiered client/server architectures and the partitioning of systems into vertical domain oriented slices (Zhao, 1998). Furthermore, the number of legacy systems that are difficult to maintain is increasing and the desire to replace these systems is growing. However, to estimate and plan the migration progress these legacy systems must be understood and documented. According to Hunold, Krellner, Rauber, Reichel and Runger (2009) these systems may have grown over decades and hence no clear understanding and definition of the system architecture and structure exists.

Documentation of the system may be scant or out of date. In these circumstances, software clustering may provide support so that developers are able to gather information, to understand the structure, decomposition and dependencies of the software system and to plan and implement refactorings and extensions of the system.

The concept of clustering has a long history in software. Parnas (1972) voiced the idea to unite low level software entities into modules. He further suggested that procedures that manipulate the same data entities should be united into one module. Each module should hide its inner design to other modules behind interfaces. The interfaces therefore form a thin communication mechanism through which other modules can access functionality. The key of this information hiding principle is that the interface does not unfold information about the internal module design. This information hiding principle makes it feasible for a developer to safely replace the implementation behind the interfaces. It can also be seen as the cornerstone for a high cohesion and low coupling design approach.

Some of the basic principles for good object oriented design have their origin in the ideas of Parnas (1972). The object oriented paradigm incorporates the notions of clustering and abstraction by uniting methods, attributes and data in classes.

Booch (1994) emphasizes the importance of using abstraction to unite similar structure and behaviour in related objects and of using encapsulation and interfaces to implement information hiding and modularization to support high cohesion and low coupling. As illustrated in the next paragraphs, nearly all software clustering approaches are based on these principles.

Software clustering results in a new level of abstraction by dividing a software system into different subsystems (Wiggerts, 1997). Through this abstraction the number of artefacts in the system will be reduced. Mancoridis, Mitchell, Chen and Gansner (1999) suggest that this reduction of artefacts reduces the system's evident structural complexity and should increase the understandability of the system design, structure and dependencies.

From its early origins, software clustering has continued to be an active area of research. Much of the more recent work has relevance to this current study. For example Lee and Yoo (2000) introduced a method which applies reverse engineering for object oriented systems. This methodology is divided into five different phases: form usage analysis, form object slicing, object structure modelling, scenario design, and model integration. This model delivers an approach to conduct reverse engineering, which adheres to the principles of the object orientated paradigm. Tichelaar (2001) proposed a framework which analyzes object orientated code dependencies. Tichelaar (2001) reflected the aspects of object-oriented systems that are relevant to reengineering. Furthermore, Tichelaar (2001) showed how a metamodel can effectively deal with differences of programming languages, such as static versus dynamic typing and single versus multiple inheritances versus *Java* interfaces. Tichelaar (2001) showed how the use of an independent core together with mappings of multiple object orientated languages provides an effective common coverage of these languages. Even though the implementation of a dependency graph algorithm is not part of this work, the paper from Tichelaar (2001) supports the process of extracting a dependency graph and is therefore relevant as a basis for the work reported here.

An approach to include external additional data in the reverse engineering approach is introduced in Xiaomin, Murray, Storey, & Lintern (2004) who combined a task concept and the logs of the version control system into the reverse engineering process. This additional information allowed the authors to identify the reasons for certain system dependencies. Based on the dependencies the correlating tasks can be identified. Furthermore, Xiaomin et. al (2004) saw a future trend in the integration of reverse engineering tools in the development context to provide architecture and dependency information to the developer. In the opinion of Xiaomin et. al (2004) this integration will help developers to orientate themselves in the system and be sensitized to the system architecture. Even if the inclusion of external data will not be relevant within this research, Xiaomin et. al (2004) hinted that the reverse engineering process can be guided by different data.

The studies of Jiang, Golf, Harman and Li (2007), Seng, Bauer, Biehl and Pache (2005) and Mancoridis, Mitchell, Chen and Gansner (1999) combined the areas of reverse engineering and SBSE and so are of particular relevance here. Jiang et. al (2007) introduced a framework for search based slicing. SBSE was here used to identify and decompose dependency structures within software systems. In addition to the development of the framework, the approach is applied within a case study. Within this case study they evaluated the application of the search based algorithms greedy, hill climbing and genetic algorithms regarding performance and similarity of results. In summary, the greedy algorithm outperformed the other algorithms in regard of the processing time to create a solution. This result is certainly based in the simplicity of the greedy algorithm. Jiang et. al (2007) did not explicitly discuss whether the quality of the cluster results of the hill climbing and genetic algorithms are better than the results of the greedy algorithm. The research of Seng, Bauer, Biehl and Pache (2005) revealed an approach to improve the subsystem decomposition of a software system. The basis of this approach was a genetic algorithm that computes a decomposition of a software system into subsystems. Seng et al. (2005) used a fitness function that incorporated heuristic data in terms of certain metrics such as *cohesion*, *coupling*, *complexity* and *cycle analysis*. These metrics are described in detail by other authors, such as Martin (1994). Additionally, Seng et al. (2005) motivated the idea that the research can be expanded with other architecture-related metrics and that weighted graphs could add further information to the fitness function. Jiang et al. (2007), Seng et al. (2005) and Mancoridis et al. (1999) depicted that the cluster analysis of software systems can be seen as a search based problem and that metaheuristic algorithms are potentially able to deliver good solutions in reasonable time. To illustrate the possible application of SBSE in the field of software engineering the *Bunch* tool was developed in the research of Mancoridis et al. (1999) and Mitchell (2002). The tool *Bunch* is able to identify clusters and display dependency graphs within software systems. *Bunch* is running as a standalone system and is not integrated into a development environment. The *Bunch* tool can process files, which are stored in the *Module Dependency Graph (MDG)*. This describes a text file, in which every line represents a dependency with a source artefact and a destination artefact. The work of Mancoridis et al. (1999) and

Mitchell (2002) are relevant sources for the implementation of this research hence they will be further examined throughout this research.

In conclusion, this section has given an overview of the development in the field of software clustering and describes relevant research in this area. Additionally, this paragraph emphasized the application of SBSE in the area of software clustering. The next section portrays the software clustering process.

2.3.2 The Software Clustering Process

The research objective of this work involves the implementation of a software component to decompose software systems. Hence it is important to create a good understanding of the software clustering process.

In the literature a variety of different clustering approaches can be found. This section examines the similarities between these clustering approaches. Wiggerts (1997) identifies three important elements that must be addressed in order to design software clustering systems. Mitchell (2002) also expressed these important elements, namely:

- Exposure of the artefacts to be clustered
- Identification and measurement of criteria to determine the similarity between the artefacts
- A clustering algorithm which applies the similarity measurement

Within the next three sections these three fundamental phases are illustrated and the important parts for this research are elucidated.

Exposure of the Artefacts

To derive a cluster landscape from a software system a sufficient representation of the software system artefacts and their dependencies to each other have to be identified. This representation depends on the requirements of the clustering approach.

Within the reviewed software clustering approaches, e.g. Mancoridis et al. (1999) and Dietrich, Yakovlev, McCartin, Jenson and Duchrow (2008), the software system is

represented as a directed graph $G = (V; E)$. The set of nodes V represent the artefacts of the software system. The edges $E \subseteq (V \times V)$ represent the relations between the artefacts. The goal of software clustering is to partition this graph into meaningful subsystems. This illustrates that software clustering can be seen as a graph partitioning problem. The graph partitioning problem has a computational complexity of NP (Mancoridis, Mitchell, Rorres, Chen, & Gansner, 1998). The number of possible solutions to cluster software graph increases exponentially with the number of artefacts (nodes) within the software system. The optimal solution of a software clustering problem cannot be found in polynomial time. Regarding this the application of heuristics to reduce the computational complexity of the software clustering problem is interesting.

Mitchell (2002) and Wiggerts (1997) described factors which they asserted need to be considered when designing the representation of the artefacts. One of the influencing factors is the desired granularity of the recovered system. This refers to the level of the clustering approach. The clustering, for example, can be applied at method, class or package level or even a multi-level approach can be required, which considers more than one artefact level. Another factor considers the attributes of the relations between two artefacts. Is it beneficial if the relations between two artefacts carry a weighting to express some form of connection strength? If a weighting is considered, there is a need to determine which attributes are of special relevance. They have to be included into the weighting. This countenances also the possibility to include multiple weights.

Identification and Measurement Similarity Criteria

After the representation of the artefacts and their relations between the artefacts are defined, the similarity between two related artefact representations has to be determined. The similarity defines the connection strength between two related objects. Higher similarity measurements indicate a stronger similarity or connection between two artefact representations.

If a multiple weight approach is chosen, consideration then needs to be given if there is a different importance between the individual weights. For example, because of the

different character of the *implements* or *extends* relationships within a Java based software system, these could potentially have a different weighting than the dependency relationship. The *implements* and *extends* relationships hint a high degree of similarity and express a high similarity and connection between these classes. Another example is the inclusion of interfaces in the dependency relationship. The interface encapsulates the specific implementation from the accessing class. This relationship should command a lower weighting than the relationship between two concrete classes.

In Muller, Orgun, Tilley and Uhl (1993) a similarity measurement called *Interconnection Strength* was used to determine the strength between two artefacts, which expresses the number of common artefacts which are accessed, exchanged or shared by two components. In Schwanke and Hanson (1994) a similarity measurement is identified which evaluates a set of features. Each feature represents a similarity measurement between a pair of artefacts. As a result the similarity feature can either exist or not exist within the two artefacts. Schwanke and Hanson (1994) presented a formula to normalise and weigh these similarity and distinction features. Constants were also included to allow a user-defined weight for similarities and distinction between artefacts.

The Cluster Algorithm

After defining the representation and similarity measurement method, the data required to cluster the dependency graph is available. Based on the representation and the similarity measurement the cluster algorithm can create a cluster landscape of the software system. The distribution of the artefacts is the task of the clustering algorithm as the central core of the software clustering process.

Mitchell (2002) stressed that most clustering algorithms arrange their clusters hierarchically. Hierarchical clustering begins with the low level artefacts (e.g. methods, procedures, classes) and organises them into subsystems. Based on these subsystems the cluster algorithm creates the next level of abstraction and clusters these subsystems based on the similarity measurements into new larger subsystems. Finally, it would be possible that only one subsystem is left, which contains all subsystems of

the previous hierarchy. As a result a tree is created comprising leaves that are the low level software artefacts and inner nodes that are subsystems. According to Mitchell (2002) this hierarchical clustering is beneficial in terms of aiding a developer to understand the structure of a software system. This hierarchical approach facilitates the developer in understanding, analysing and revising the software design at different abstraction levels and supports the ‘fading out’ of irrelevant system components. Schwanke and Hanson (1994) proposed a hierarchical algorithm that combines the two artefacts with the highest similarity into one new subsystem until only one subsystem is left. This offers the user a variety of different solutions on different abstraction levels. A disadvantage of the approach is that some subsystems evolve more quickly than others. Certain system parts may evolve into higher subsystem levels during the first iteration while other system parts are still on a low artefact level.

The clustering of a software system in a hierarchical manner is inappropriate for this project. It does not align with the objective of this research. The aim of this research is to present clustering results which feature the same evolution level. The asymmetry of the hierarchical cluster development is contradictory to this requirement. However, the idea to apply an iterative cluster process and evolve the cluster quality during the multiple cluster cycles is promising. For this, the inclusion of user knowledge into these cluster runs to direct the clustering results is a promising approach that will be discussed in section 2.3.3. But this user influence does not lift the clustering onto a new abstraction level but rather redirects the cluster process to improve the cluster quality. An additional approach, which is driven by the hierarchical clustering is the clustering on abstraction levels. The hierarchical clustering approach from Schwanke and Hanson (1994) produced the basis for the next clustering results from the previous clustering results. But as mentioned in section 2.1.1, current software systems exhibit already mechanisms to abstract software systems. For example: *Java* based software systems merge methods into classes and classes into packages. Current cluster approaches often work on file or class level (Anquetil & Lethbridge, 1999a; Jiang et al., 2007; Seng et al., 2005). The work of Dietrich et al. (2008) introduced the *Barrio* component, a non search based approach which is able to cluster *Java* systems on class level. The *Barrio* tool partitions a graph, based on the smallest path between two

sub graphs utilizing the *Girvan-Newman* algorithm (Girvan & Newman, 2002). The user can control the separation level, which determines the maximum strength between two sub graphs. Depending on the separation level the clustering produces different cluster landscapes. As the clustering relies on the weakest path between two sub graphs, the *Barrio* tool can give good indications for the decomposition of the system. Additionally, the cluster result can be compared with the real package classification.

As the approach of Mitchell (2002) reads an input file with the dependency information, it is independent of the application level. But it cannot utilise any special characteristic of the dependencies or different levels of the analysis. Aligned with the research objective of this work to reduce the complexity of software systems, the application of a cluster approach, which is able to cluster also on higher abstraction, is beneficial on higher abstraction levels e.g. package level. Certainly it is important that the system commands about a system wide high functional cohesion on the clustered artefact level. Would this not be the case the cluster algorithm would be misled by the multiple connections of the artefact.

Muller et al. (1993) introduced a cluster approach, which includes an artefact into a cluster, when the similarity measure (*Interconnection Strength*) extends a threshold. This threshold can be defined by the user. If an artefact exceeds these thresholds with two relating artefacts all three involved artefacts are combined in one cluster. A disadvantage of this approach is the missing transparency of the *Interconnection Strength* for the user. The introduction of thresholds to control the clustering will give the user the chance to further adapt the result to his personal preferences. As mentioned before the transparency of the impact of the threshold is also of importance.

Anquetil and Lethbridge (1999b) illustrated the recovering of a software architecture based on the similarity of file names. The artefact representations are labelled based on established file abbreviations for source code files (e.g. “domain”, “material”, “tool”) and name similarities. Based on this approach, a heuristic-based cluster algorithm assigns the artefacts into clusters. Anquetil and Lethbridge (1999b) suggested a variety of different heuristics to distribute the artefacts into clusters, but

favoured a heuristic that assigns artefacts into clusters based on the first name abbreviation. The disadvantage of this approach is that if a software system is not built with a strictly followed naming policy then it is not applicable. Furthermore, if a particular file name does not follow the name policy, it is unlikely that it will be assigned into the correct cluster. In Mitchell (2002) three clustering algorithms (exhaustive, hill-climbing, genetic algorithm) were pioneered to cluster software systems into subsystems. The goal of these heuristic cluster algorithms is to maximize an objective function. Mancoridis et al. (1999) introduced the term Module Quality (MQ) as a description for the quality of a cluster solution. In Mitchell (2002) the two objective functions *BasicMQ* and *TurboMQ* were introduced. Each of these fitness functions rewards high cohesion within a subsystem and penalizes high coupling between subsystems. The *BasicMQ* fitness function measures the *inter-connectivity* as the connection between the artefacts of two distinct clusters and *intra-connectivity* as the connections between the artefacts of the same cluster. Based on the measurement of the *inter-connectivity* and *intra-connectivity* for every cluster is the tradeoff between *inter-connectivity* and *intra-connectivity* calculated by rewarding the existence of highly-cohesive clusters and the penalization of too many inter-edges. Mitchell (2002) asserted that the *BasicMQ* fitness function delivers good results but also involves a high computational complexity. Respectively, the *TurboMQ* measurement was introduced. The *TurboMQ* measurement features a computation complexity of $O(|E|)$, where E represents the set of edges of the analyzed graph. *TurboMQ* fitness function measures a *ClusterFactor* for every cluster of the partitioned graph. It is calculated for every cluster of the partitioned graph. The *ClusterFactor* is defined as a normalized ratio between the total weight of the internal edges (edges within the cluster) and half of the total weight of the external edges (edges that exit or enter the cluster). The half weight is applied as the external dependencies are connected with two clusters and respectively penalizes two clusters. Mitchell (2002) contemplated that the *ClusterFactor* could be calculated by the application of weighted edges. This approach was not further examined and evaluated by Mitchell (2002) or continuative work as (Mitchell & Mancoridis, 2006, 2008). In regard to the performance the genetic algorithm introduced in Mitchell (2002) is able to identify

better solutions than the hill climbing algorithm. This observation is independent of the *BasicMQ* and *TurboMQ* fitness function. On the opposite, the hill climbing algorithm outperforms the genetic algorithm in computational time.

In conclusion, this section has illustrated relevant research in regard to the clustering process of software systems. The illustration of the research followed the classification of Wiggerts (1997) into the three main fields exposure of artefacts, similarity measurement and the cluster algorithm. Further on an aim of this research is to investigate the performance and quality of the clustering by including expert and domain knowledge into the clustering process. Regarding this, the next section gives an overview about relevant research, which enables the inclusion of expert and domain knowledge into the clustering process.

2.3.3 Inclusion of Expert Knowledge

This section discusses relevant research in the field of expert and domain knowledge inclusion into the cluster process.

The software clustering approaches can be distinguished in manual, semiautomatic and fully-automatic clustering (Mitchell & Mancoridis, 2006). Manual clustering approaches supply the user with a framework to implement the clustering, but do not support the user with any suggestions or help to partition the software system automatically. On the contrary fully automatic approaches do not allow the user to influence the cluster process. Mitchell and Mancoridis (2006) stressed that semi-automatic cluster approaches support the user with automatic functions, but also require manual user interaction to conduct the software partitioning. According to Mitchell and Mancoridis (2006) the clustering approach introduced by Mitchell (2002) is a fully automatic approach, which allows the user to manually supply additional cluster information. However, the *Bunch* tool only allows the user to add additional knowledge about the structural information in the format of a text file. This information is read once and included at the beginning of the cluster process. *Bunch* does not support the interactive inclusion of user data in an iterative or incremental manner.

The *Bunch* tool produces different cluster results with the same user input data. Mitchell (2002) argued that because of the heuristic search approach a deterministic solution is not possible. This behaviour may be confusing for the user – especially if he or she is not familiar with stochastic processes - and so instead anticipates a precise manipulation of the solution. In conclusion, non-deterministic behaviour of the cluster algorithm would corrupt an interactive approach.

2.3.4 Evaluation of Software Clustering Quality

This section demonstrates approaches to evaluate software clustering solutions. The evaluation of clustering solutions features a variety of challenges. The challenges are the compliance of the objectivity, the determination of the relevant aspects to evaluate a cluster solution and the consideration of the aim of the clustering.

Most research in the area of software clustering promotes the use of experts to evaluate their clustering approach (Mitchell & Mancoridis, 2001). This approach can deliver a high quality evaluation, but relies on the subjective perceptions of the expert. In addition, the expectations of expert effort and necessary knowledge to evaluate a cluster solution are drawbacks of this approach. This motivates the desire to find a more formal way to evaluate cluster solutions.

A distance metric for software clustering was introduced by Tzerpos and Holt (1999) to evaluate the similarity of two software system decompositions. The research assumed the existence of an expert decomposition against which to validate the cluster approach decomposition. Anquetil and Lethbridge (1999b) adopted an approach to compare different similarity measurements to estimate the quality of a clustering approach. Meanwhile, Mitchell & Mancoridis (2001) introduced the CRAFT framework, which offers a more formal way to evaluate and compare clustering approaches. The CRAFT framework uses the hillclimbing and genetic algorithm from the *Bunch* tool. It is also possible to add individual algorithms to compare the results with these algorithms. The idea of the research of Mitchell and Mancoridis (2001) was that the intersection of all algorithms is likely to deliver a good basis on which to evaluate any individual algorithm.

The validation basis for a cluster landscape evaluation is created by multiple runs, illustrating that the CRAFT framework aims especially for non deterministic algorithms, which create different cluster outputs. Based on these multiple runs the CRAFT framework calculates the distribution for each artefact. A possible result for one artefact could be that artefact *A1* is combined with artefact *A2* in 90% of the evaluation runs and combined with artefact *A3* in 10% of the evaluation runs. This outcome can be compared with the results of another cluster algorithm and an aggregate outcome can be determined.

In such a scenario it has to be considered that the included cluster algorithms should take into account the same software attributes. Otherwise the outcomes would not be comparable in any valid way. Considering the possibility to visualize a software system from different perspectives the evaluation should also consider the perspective criteria to evaluate the clustering result. If the cluster algorithms within the CRAFT framework do not focus on the same perspective criteria the evaluation with the *CRAFT* framework would be meaningless.

2.4 Software Metrics

This section introduces the field of software metrics. Beginning with the theory of measurement the motivation for the application of software metrics in SBSE is drawn. Additionally different classes of software metrics are demonstrated and it is stated which metric class is suitable for this research. Further on, recent research in the area of software metrics, with special focus on software clustering and SBSE, is examined.

The measuring theory model of metrics is based on the empirically relational system and on the numerically relational system (Fenton & Pfleeger, 1997). An empirically relational system exists of an amount of objects of the reality. Objects can be physical objects, events or also abstract things. Examples of objects are people, programs, documents or test phases. Attributes are qualities of objects which can be, for example, the size of a person, the circumference of a document or the complexity of a program. This illustrates that an object itself cannot be measured but the attributes, which describe the characteristic of the object, can be measured.

On account of the different attribute values an order originates between the quantities of objects. The objects have an order in relation to an attribute. The relation shows an ordinal criterion for the objects. Because of the relation an order is defined between two objects. Empiric relations can be for example, "is longer" or "is more structured than". A mapping of concrete values does not exist in the empirically relational system. The difficulty of classification of an empiric relation is applied, for example, in the relation is "more complex than" to an amount from software programs. These empiric relations between two software programs cannot be ascertained objectively. With the help of a relation the empiric order is based on observations, comparisons and appraisals of the reality. On the other hand the numerical relational system defines formal objects (e.g. figures) and exact relations (e.g. ">"). The purpose of a measurement is the transformation of informal (empiric) objects to formal objects. The aim of a transformation from a relational system (empirically to relational system) in another (numerically relational system) is to express the empiric order within the numerical system.

This transformation is the task of a metric. A metric is a surjective function $f: S \rightarrow \mathbb{R}$, which transforms an attribute set S into the numerical relational system \mathbb{R} (Schneidewind, 1992). Based on this formal definition of a metric, and with reference to section 2.2.2, the similarity of the fitness function and a metric can be recognised. This link was also drawn in the research of Harman and Clark (2004) that illustrates the similar characteristics of metrics and the fitness functions and motivates the possibility to include different metrics into the fitness function. Thus the work of Harman and Clark (2004) constituted the base to apply further software metrics in the field of software clustering and SBSE.

The purpose of metrics is the measurement of certain software qualities (Erhard, 1991). These qualities cannot be measured by metrics directly. A metric measures attributes of an artefact which gives an indication for certain quality attributes (El-Wakil, El-Bastawisi, & Boshra, 2004). Considering this, a metric is rather an interpretation of a software quality attribute and does not express the quality

attribute itself. The aim of metrics is to transfer these measured attributes into a comparable scale (Schneidewind, 1992).

An example of an easy-to-understand metric is the Lines of Code (LoC) metric. The LoC metric counts the total number of code lines within an artefact usually without counting the empty lines (Erhard, 1991). A high LoC value can be an indication of a very complex class or even a god class (Riel, 1996; Smith & Williams, 2000), which comprises a high amount of functionality and breaks the principle of encapsulation (Gamma, Helm, Johnson, & Vlissides, 1995). However, it may also identify GUI artefacts, which tend in general to be longer than service or domain artefacts. Use of this metric could help the developer to identify problematic code segments, but it would still be an individual decision if an artefact has to be refactored or not.

Considering this, a metric-based analysis should rather consider different metrics to identify problematic artefacts. A collection of metrics is called a metric suite (Hitz & Montazeri, 1996). If multiple metrics of a well combined suite produce high metric values for a certain artefact a more detailed analysis of this artefact would likely be necessary.

2.4.1 Classification of Software Metrics

To isolate software metrics, which are suitable for the work reported here, the division of software metrics by Fenton and Pfleeger (1997) into process, product and resource metrics is briefly described. This subsection provides a short overview over these three metric groups. The commentary on product metrics has greater depth, given their greater relevance to this research project.

Process metrics concern the development process associated with a software-system. Examples are the number of *mistakes/week*, expenditure estimates or indicators of the project progress. *Resource metrics* measure the available or expended resources. Examples are the utilization of personnel resources or computer resources. *Product metrics* reflect aspects of the artefacts that comprise the software product. These metrics characterize the specification, the design or source code attributes of a software product. Examples are the lines of code (LoC) metric or the number of

methods/class. A further division of product metrics into traditional and object-oriented metrics can be found in the literature (Chidamber & Kemerer, 1994). The use of traditional metrics is not purely bounded on procedural programming environments. However, the use of traditional metrics within object-oriented systems should be undertaken with caution as within the object-oriented paradigm some factors are added, which are not considered by traditional metrics. These factors have an impact on the complexity of the system and need to be included in the metric analysis of an object-oriented system. These object-oriented criteria are abstractions in classes, inheritance, packaging and polymorphism. The application of a traditional metric in conjunction with these OO-specific metrics, as for example the *Lines of Code* (LOC) metric, can be meaningful within an object-oriented system. Apart from these distinctions, a classification of metrics according to different degrees of abstraction is possible. In this case a classification can be made into micro-level and macro-level metrics. Macro-level metrics consider characteristics on the level between modules (inter-module level). The investigation of micro-level metrics is bounded on internal module characteristics. This classification is very rough, because inter- and intra-module communication can take place on different artefact levels. A further partitioning of metrics according to the application level is also possible. For example this classification could be, according to the object-oriented paradigm, a partitioning on system -, subsystem -, classes and method level. (Kiebusch, Franczyk, & Speck, 2005). These examples illustrate that many different classifications of metrics can be found in the literature. The most important point before introducing the use of metrics in relation to a software system is to determine if the metric is appropriate and useable in the environment and on which artefact level the metric analysis is considered to be most reasonable.

2.4.2 Application of Software Metrics in SBSE

This paragraph illustrates the application of metrics in relevant research projects and, with regard to the research objective of this work, focuses on the application of metrics in the area of SBSE and software clustering.

As stressed by Harman and Clark (2004), the application of metrics in search based software engineering and software clustering is promising. Harman and Clark (2004) illustrated the similar characteristics of metrics and the fitness functions and motivated the possibility to include different metrics into the fitness function. Anquetil and Lethbridge (1999b), Mitchell (2002), Seng et al. (2005) and Jiang et al. (2007) have all included metric values into the fitness function for different clustering approaches. Anquetil and Lethbridge (1999b), for example, the similarity of names as a ratio value and used this information to rebuild the structure of a software system.

The research of Mitchell (2002) was based on the measurement of cohesion and coupling measurements. This cohesion and coupling measurement was based on the measurement of the dependencies. Mitchell (2002) did not feature a difference strength or characteristic of the dependencies.

Seng et al. (2005) and Jiang et al. (2007) introduced search based clustering and slicing approaches, which were using software metrics to guide the search process. Seng et al. (2005) included the cohesion, coupling, complexity, cycles, and bottleneck metric into the fitness function. The work of Jiang et al. (2007) focused on the slicing of software systems and applied the *coverage* and *overlap* metrics, which were inspired by the slicing approaches from Ott and Thuss (1993). As illustrated in section 2.2 and 2.3 the research of Anquetil and Lethbridge (1999b), Mitchell (2002), Seng et al. (2005) and Jiang et al. (2007) concentrates on the application of the hillclimbing and the genetic algorithm metaheuristic.

In conclusion, little research has been conducted in the overlapping fields of software metrics, SBSE and software clustering. The research of Mitchell (2002) and Anquetil and Lethbridge (1999b) concentrated on the application of one measurement to derive a cluster landscape. On the opposite, the research of Seng et al. (2005) and Jiang et al. (2007) applied a set of metrics to offer a decision base for the fitness function. But the weighting of these metrics in combination with each other is fixed. Seng et al. (2005) and Jiang et al. (2007) both applied the same weighting for all metrics and did not allow the changing of the weighting of these metrics. The research of Mitchell (2002), Seng et al. (2005) and Jiang et al. (2007) was able to identify different solutions by

changing the applied fitness function or the metaheuristics. These solutions differ merely because a better or less feasible solution is found within the search space. This illustrates that the exploration of the search space is influenced by the fitness function, but the criteria which define a good or bad solution do not change. Mitchell and Mancoridis (2008), Seng et al. (2005) and Jiang et al. (2007) motivated that a configurable fitness function would be interesting and a field for further research.

2.5 Summary

This section concludes the literature review in the areas of software architecture, SBSE, software clustering and software metrics. Based on the literature review of the studies the limitations and challenges can be derived to develop the research questions of this research.

In particular, the work of Jiang et. al (2007), Seng et. al (2005) and Mitchell (2002) constitute the main basis for the present research. These research projects combine the areas of SBSE and reverse engineering and encourage the further exploration of SBSE in the field of software clustering. A relevant finding within the software clustering literature review is that it seems to be at least beneficial, to include additional user data and expert knowledge to the clustering process. To date user input has been limited to the application of thresholds and the provision of an initial user data set which is considered by the cluster process (Mitchell, 2002). A user directed guiding of the clustering process, where the user is able to contribute additional user data into the cluster process to influence the search process and its outcomes would be possibly beneficial and increase the likelihood of identifying good solutions. Integration of such support into a development environment should simplify the orientation of developers within a given software architecture and rapidly provide information regarding the dependency structure. A developer can then obtain the necessary information for further modularization and equalization of the software system.

Mitchell (2002) depicted the inclusion of one dimensional weight into the dependency graph. Furthermore, the work of Seng et. al (2005) and Jiang et. al (2007) portrayed the possibility to include weighted graphs into a fitness function to identify important

dependency structures. Harman and Clark (2004) illustrated the similarity of the fitness functions and software metrics. So far research was limited to the application of cohesion and coupling, name similarity, cycle and bottleneck metrics. There is potential, then, to explore further software metrics for inclusion into the fitness function. In Seng et. al (2005) the combination of multiple metrics was applied. However, Seng et. al (2005) did not envisage a controlled manipulation of the inclusion strength of these metrics. Mitchell (2002) stressed that the inclusion of multiple metrics would be interesting to examine a software system from different perspectives. It appears that to date no research has been conducted to recover the structure of a software system to create different cluster landscapes of the implementation perspective. Considering this, the inclusion of multiple weights in a fitness function would seem promising to support multiple implementation perspectives and to identify problematic code segments (code smells).

The application of metaheuristics in software clustering appears to be limited to the application of hillclimbing (Jiang et al., 2007; Mitchell, 2002), genetic algorithms (Jiang et al., 2007; Mitchell, 2002; Seng et al., 2005) and the greedy algorithms (Seng et al., 2005). Nevertheless, Seng et. al (2005) applied the greedy algorithm in the similar field of software slicing and not directly in the field of software clustering. None of the previous research applied tabu search to the challenge of software clustering. Tabu search with its deterministic character and its efficient memory exploration model (Sung & Jin, 2000) is certainly promising for the application in software clustering.

Considering the challenges and limitations of current studies and the opportunities for promising research paths the final research objectives can be stated.

3 Research Objectives and Methodology

Based on the outcomes of the literature review and the discussion of promising research opportunities, this chapter states the research objectives, identifies the research questions and thereon describes the methodology and design of this research.

3.1 Research Objectives

The aim of this research is to determine if a user-directed search-based software clustering approach is applicable to support stakeholders with information about the structure and dependencies of a software system. This aim has arisen from the literature review, with the outcomes and limitations of prior work leading to an assumption that a user directed clustering approach, which would give the stakeholders more control over the clustering process, has the potential to contribute to the quality of software clustering.

Regarding the previously applied metaheuristics in the field of software clustering, the application of tabu search in the area of software clustering is adopted here. The flexibility and possibility of parameterisation of the tabu search algorithm should align with the requirements of a user-directed software clustering approach. Additionally, the application of the greedy algorithm as a comparison and evaluation baseline is intended, building on the research of Jiang, Gold, Harman, & Li (2007) who applied the greedy algorithm in the similar field of software slicing.

A further aim of this research is to investigate if the inclusion of multiple metrics in the search fitness function enables the user to create different implementation perspectives of a software system.

Based on these formulated research objectives the research questions for this work are derived.

Research Questions

The following research questions are answered within this research:

- Can a user directed and semi-automatic clustering approach contribute to the quality of software clustering?
- Is tabu search applicable in the area of software clustering?
- Does the inclusion of multiple metrics in the fitness function enable the clustering of a software system into multiple implementation perspectives?

Based on the research objectives and the identified research questions the research methodology is now described.

3.2 Research Methodology

In order to address the research questions in a robust and rigorous way and to demonstrate and evaluate the effectiveness of the approach adopted, an appropriate methodology needs to be selected. As this research is both exploratory and constructivist in nature it therefore utilises the system development research methodology (SDRM) of Nunamaker et al. (1990) in accordance with the design-science research guidelines introduced by Hevner et al. (2004) to specify, design, develop and evaluate a software component to exploit search based principles in the analysis of the dependencies within a software system. Constructive methodologies (Jones, 2004; Nunamaker & Chen, 1990), comprising the main steps of building and evaluating an artefact, is commonly employed in software engineering and information systems research. Figure 3.1 depicts the individual stages of the SDRM research process. Because of the explorative character of systems development research projects the SDRM research methodology does not demand a strict sequential process. Findings in later stages can mean that previous stages have to be revisited. Each stage features activities to reach certain goals. The next section describes the design of this research applying the SDRM.

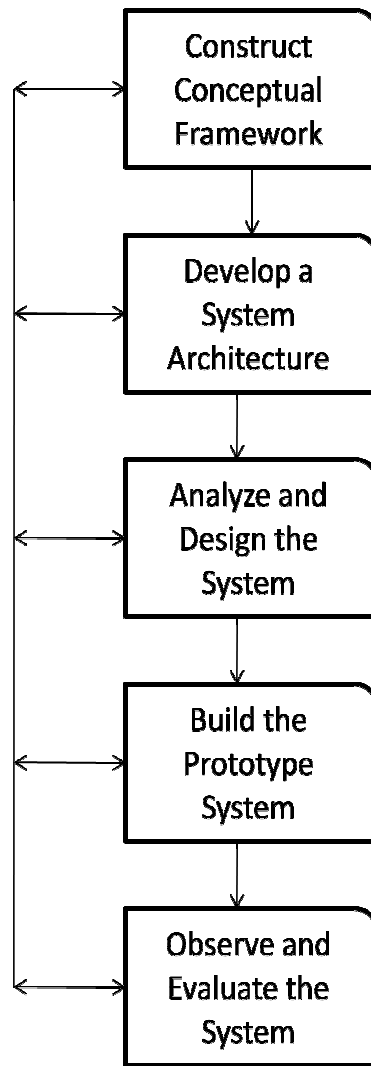


Figure 3.1 : Research process of the SDRM (Nunamaker & Chen, 1990)

3.3 Research Design

Based on the research objectives, the resulting research questions and the selection of the SDRM research methodology, the design of the research is as follows.

Construct a Conceptual Framework

To employ a constructive paradigm without fully understanding the problem space could potentially lead to inconclusive research or misleading results. This research is founded on an in-depth literature review in the area of software architecture analysis, software metrics, software clustering and search based software engineering. The research aim and research questions have been derived based on the consideration of promising paths and a discussion of the placement of this research relative to prior work.

Design and Implementation

With the context of the research defined, a constructive methodology lends support to the design and implementation of a component that addresses the research aim and the identified research questions. Within this research this phase comprises the steps and goals of the “develop a system architecture”, “analyse and design the system” and “build the prototype” stages of the SDRM. The scope and functionality of the developed Search Based Reverse Engineering (SBRE) component are defined as follows.

One research objective of this work is to examine if tabu search is applicable in the field of software clustering. This demands that a specific implementation of the tabu search metaheuristic, with its ideas and concepts, is applied within the SBRE component. However, the framework should not be bounded only to the application of tabu search. An implementation of the greedy algorithm should also be implemented to provide a basis for comparison and evaluation of the tabu search. More generally, the SBRE component should be built so that it supports the inclusion of additional metaheuristics.

An examination of the effectiveness of applying multiple metrics in positively influencing clustering outcomes is another important element of this research. In particular, the use of multiple metrics should enable the illustration of different implementation perspectives and the identification of *code smells*. This requires that a framework is developed which supports the flexible inclusion, extension and exclusion of metrics into the clustering process. To enable this flexible configuration of metrics the user has to be able to configure the inclusion of every metric individually.

The main objective of this research is to determine if a user-directed approach can contribute to the quality of software clustering. During the literature review and the identification of limitations of prior research four approaches that give the user more control over the cluster process have been identified. The first approach aims to increase the efficiency of the clustering process through the inclusion of user knowledge in an interactive manner. This can guide the search into more promising areas of the search space. The user can add constraints into the clustering process

interactively and is able to trigger the clustering process for the recalculation of the software clustering. An additional user-directed clustering mechanism is intended to deliver a reduction in the complexity of the software clustering by transposing the clustering to a higher level of abstraction. Programming languages offer different levels of abstraction e.g. *classes* and *packages* in *Java* systems. It has to be examined, then, if it is beneficial that the system is clustered at higher abstraction levels than class level. To evaluate this, the search based software clustering system has to be able to cluster software systems at various abstraction levels.

Hence the applicability of the flexible multiple metric approach and the tabu search algorithm are to be examined to determine if these approaches contribute to the quality of software clustering. The tabu search algorithm features a variety of influencing parameters e.g. number of maximum tested solution candidates, length of the tabu list and the triggering of the intensification and diversification method. The component should support the flexible configuration of these parameters to enable the examination of the contribution to software clustering quality.

Evaluation

The SDRM requires the observation and evaluation of the constructed component to gather the necessary data to answer the research questions. To enable the evaluation of the SBRE component, an experimental methodology is applied to assess both utility and performance (Collis & Hussey, 2009). The experimental methodology can confirm a theory, examine a relationship, evaluate the accuracy of a model or validate a measure (Collis & Hussey, 2009). To conduct the experimental methodology an expected outcome has to be determined. Additionally, the environment and the procedure have to be defined to enable the reproduction of the experiment. After data collection, the results are interpreted and compared with the expected outcome. The design of the experiment should be aligned with the research objectives in order to ensure that the necessary data are collected and thus the research is able to answer the identified research questions. With regard to the research questions stated above, the next section illustrates the requirements to collect a sufficient data base.

The main research question of this research addresses the contribution of a user-directed approach to the quality of software clustering. The term 'quality of software clustering' has a variety of interpretations. For this research, the influence of user interaction on the feasibility of achieving cluster solutions and the effect of interaction on the complexity of the software clustering process are relevant software quality dimensions. A collection of experiments, which are described in Chapter five, contribute data to this aspect. An assessment of the technical applicability of the inclusion of user knowledge into the clustering process and an interpretation of the effects of the clustering result are given in section 5.4.3. The other approaches incorporated in the user-directed clustering strategy adopted here are considered as part of the evaluation of the additional research questions. A final discussion is provided in section 6.1.3 to address the research question and to determine if a user directed clustering and semi-automatic clustering approach can contribute to the quality of software clustering.

The evaluation of the tabu search algorithm includes an analysis of the applicability, performance and limitations of tabu search within a software clustering approach. The applicability of the tabu search implementation is assessed via analysis and interpretation of the software clustering results. The corresponding experiments are illustrated in section 5.3. In particular, attention is focused on the effect of the tabu search algorithm on the fitness function measurements or the computational time required to create a solution. The implementation and results of these experiments are illustrated in section 5.3.2. These data are interpreted and compared with the results of the implemented greedy algorithm, which is evaluated in section 5.3.1. A final discussion regarding the applicability of the tabu search algorithm is given in section 6.1.1.

This research considers the extraction of different implementation perspectives by including a flexible multiple-metric approach in the fitness function. The intent is that metrics can support the identification of different implementation perspectives depending on specific metric configurations. As a result, created solutions should vary depending on the configuration and should reflect the intent of the corresponding

metric configuration. Additionally, other tests are designed to assess whether a multiple metric approach is able to support the identification of *code smells* at an abstract level. A final discussion and answer to the associated research question is given in section 6.1.2.

Based on the description of the design of this research, the design and implementation of the component to enable the achievement of the research objectives is now described.

4 Design and Implementation of the SBRE Component

Driven by the objectives and the selected design methodology of this research, this chapter illustrates in detail the design and implementation of the Search Based Reverse Engineering (SBRE) component. The SBRE component embodies the designed and implemented artefact of this research. Because of the explorative character of this research and the necessity of reviewing design decisions based on findings during the development process (as is common in contemporary iterative approaches), the design and implementation processes are consolidated in one chapter. This chapter defines and describes the architecture, the components and the algorithms. It constitutes the end products of this research endeavour to enable the user to control the clustering process by the configuration of metrics and adding domain knowledge.

The overall aim of the present work is to help the developer and architect in arriving at an optimal structure for a software product. This emphasizes the development of a software clustering framework which applies a metaheuristic guided search process.

As mentioned in section 2.3.2, and introduced by Wiggerts (1997), a clustering framework comprises the three phases: exposure of the artefacts to be clustered, identification and measurement of criteria to determine the similarity between the artefacts and a clustering algorithm which utilises the similarity measurement. For the design of the system architecture of this component, this work allocates the selected approach for software clustering systems into these three phases. Referring to the research objective, this research project will employ the tabu search metaheuristic. The special requirements for the design of metaheuristics are covered in the algorithm design section. As described in section 2.2.2, a framework for the design of this metaheuristic is provided by Talbi (2009). Talbi (2009) subdivides the metaheuristic design process into three phases: design of the solution representation, design of the fitness function and design of the constraint handling.

Before the design and implementation of the individual components of the SBRE component are described, the next section outlines the development environment used to support this research.

4.1 Development Environment

Given the research goals and the benefits of the integration of components into one or more IDEs, this component is intended to be developed as an integrated development component. In the present research the SBRE component is to be developed within the Eclipse plug-in framework, due to the easy deployment process and modular architecture of this particular framework. As a consequence the analysis will be bound to Java systems only. While this represents a limitation on the applicability of the work, it is appropriate to fulfil the proof of concept- and prototype-character of this project.

The design, implementation and evaluation should be guided by the identified research questions and the stated research scope. Also taking into account the limited time and resources of the project, it is meaningful and appropriate to utilise external components to promote the project quickly into an applicable prototype stage. The early application of a prototype will allow an early evaluation and consideration of the applicability of the SBRE component.

It is evident that a software clustering framework is needed to answer the formulated research questions. An appropriate candidate for this project is the *Barrio* clustering component introduced by Dietrich, Yakovlev, McCartin, Jenson, & Duchrow (2008). The *Barrio* component is an *Eclipse* plug-in and the source code is available at the project home page (Dietrich, 2009).

4.1.1 The *Barrio* Framework

The *Barrio* framework comprises the three necessary phases for cluster analysis mentioned by Wiggerts (1997). For the source code analysis the Code Dependency Analyzer (CDA) framework is used (Duchrow, 2009), which creates a dependency graph and stores it in the open *Object Dependency Exploration Model (ODEM)* format. The output in ODEM format is transferred into the internal *Barrio* graph representation. The *Barrio* framework uses the *Jung* graph library (O'Madadhain, Fisher, & Nelson, 2009) as internal graph representation. Based on the exposure of the artefacts, the classification and clustering analysis are executed. Finally, the graph including the clustering information is displayed. For the visualization the prefuse graph system

introduced in Heer, Card, & Landay (2005) is used. The clustering process of the *Barrio* framework is illustrated in Figure 4.1.



Figure 4.1 : *Barrio* software clustering process

The *Barrio* framework does not support search based algorithms in the classification process. Given the aims of this research project, the classification and clustering component of the *Barrio* framework had to be replaced to apply a search based clustering approach. Additionally, the visualization component had to be replaced to address the research questions of this research related to incorporating a greater element of user direction. However, the *Barrio* component contributed positively to this research, especially in relation to the transformation of the Java source code into a representation suitable for the similarity measurement, classifying and clustering process. In retrospect, the *Barrio* framework provided an initial infrastructure that enabled this work to achieve a quick project start-up, but during the evolution of this research project a majority of the *Barrio* components had to be discarded or replaced.

4.1.2 The *Bunch* Framework

An initial idea was to include the *Bunch* framework introduced by Mitchell (2002) in the *Barrio* software clustering component, since the *Bunch* framework applies heuristic search approaches. The *Bunch* framework can modularize software systems by applying Hill Climbing and Genetic Algorithms as clustering algorithms. Furthermore, the *Bunch* framework offers an extension API to integrate independently developed search-based algorithms.

Unfortunately, a number of drawbacks made the application of the *Bunch* framework within this project impractical, meaning that it would not be possible for the outcomes to address the formulated research questions. The *Bunch* system is only available as a binary jar file. Thus, no source code of the *Bunch* tool is available to conduct source code analysis and modifications within the *Bunch* tool.

As the *Bunch* framework uses the MDG file to load the artefact and dependency information and no cluster information is stored in the MDG file, an iterative and incremental clustering approach is not possible with the *Bunch* framework. The recommended source analysis tools (*Chava* and *BAT*) to create the MDG graph are no longer available or supported. This further complicated the application of the *Bunch* tool.

Mitchell & Mancoridis (2006, 2008) stated that the *Bunch* framework can consider graphs with weights. The *TurboMQ* fitness function in combination with the genetic algorithms provides this capability. However, it is not possible to ascertain whether an integration of metric values would have been possible or if an interactive and incremental clustering approach could be employed, as only on the basis of an algorithm extension would it have been possible to evaluate these changes.

The utilisation of the *Bunch* tool in an integrated approach is theoretically possible according to the API specification. However, the *Eclipse* class loader concept does not harmonize with the *Bunch* object instantiation mechanism. This caused *class not found exceptions* because of incorrect class loader assumptions within the *Barrio* source code. This problem could not be solved even in collaboration with the *Bunch* developers. It became evident that integration into the *Eclipse* (IDE) framework is not possible without major changes within the *Bunch* software system. This prohibited the use of the *Bunch* tool in terms of enabling an integrated approach.

In conclusion, the *Bunch* framework is not applicable to address the research objective within this project and so a novel search-based cluster framework had to be developed. That said, the *Bunch* framework provided a useful benchmark for the evaluation of this research.

The next sections describe the design process of this component following the design framework of Wiggerts (1997).

4.2 Exposure of the Artefacts

As stated in section 4.1 the exposure of the artefacts is conducted by the *Barrio* tool. During this research project a component is extracted from the *Barrio* tool, which

allows the user to select one *Java* project based on the existing *Java* projects in the current *Eclipse* workspace. Figure 4.2 illustrates the interface of the project selection component.

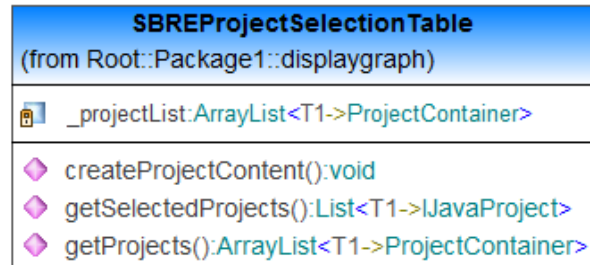


Figure 4.2 : UML class diagram of the SBRE ProjectSelectionTable

The component detects the existing *Java* projects of the *Eclipse* workspace. The component is implemented as a table with one column for the project selection status and one column for the java project name. Figure 4.3 depicts the visualisation of the *Java* project selection component. Based on the selection the component is able to return the selected *Java* project as an instance of the *IJavaProject* interface.

Java Projects	
On/Off	Project Name
<input type="checkbox"/>	sbre
<input checked="" type="checkbox"/>	testProject

Figure 4.3 : Visualisation of the project selection component

The SBRE component hands the chosen *Java* project on to the CDA component and the artefact extraction process is triggered. As a result, a directed graph is returned. The graph is an instance of the *Jung* graph framework introduced in O'Madadhain, Fisher and Nelson (2009). It takes the form of a set of nodes, which represent the artefacts of the selected *Java* project, and a set of edges, which represent the dependencies between the nodes. The *Jung* graph instance can be seen as the result of the artefact exposure process.

4.3 Design and Implementation of the Similarity Measurements

Based on the exposure of the artefacts the similarity between the extracted artefacts has to be determined. The basis for the similarity measurement is the extracted *Jung* graph instance. The result of the similarity measurement forms the decision basis enabling the clustering algorithm to decompose the software system into clusters.

This section therefore describes the design of a framework, which determines the similarity of artefacts. As derived from the literature and formulated in the research objective, the similarity measurement is implemented by the application of multiple software metrics. This requires the measurement of multiple aspects of the artefacts, with each similarity measured by the application of one suitable metric. The aim of this research project is not to recommend novel metrics or to evaluate whether certain metrics are more suitable than others for the application to the task of software clustering. It is rather the aim of this research to examine if a multiple metric approach delivers the possibility to generate and visualise different perspectives of a software system. Considering this, the selection of metrics should vary regarding their measurement goals. Additionally, the metrics should impose low computational complexity, be easy to understand and easy to implement. The inclusion of additional metrics may be appropriate if the concept has been proven as feasible. One aim of this research is to apply software clustering at the package level of *Java* programs. This requires that the metrics are applicable at different artefact levels. Depending on the focus of the clustering, some factors have to be considered, which could hinder the aim of the clustering. These are the identification of logical or functional subsystems and the existence of a dependency flow within the system. Dependency flow, describes the dependency direction of the system from higher to lower layers or subsystems in the architecture e.g. visualisation components access domain or base functionality. If the clustering algorithm simply follows this dependency flow the clustering of functional connected subsystems would not be possible. This could be overcome by considering attributes, which indicate functional connections between artefacts e.g. the rewarding of interface and inheritance relationships.

Considering the results of previous research the cohesion and coupling of artefacts (Mitchell, 2002) and the similarity of names measurements (Anquetil & Lethbridge, 1999b) have already delivered good results and thus were also utilised within this research. Each of these metrics quantify the relationship between two artefacts. Metrics of this nature have shown to deliver a good decision basis for the clustering process. In addition, there is likely to be complementary benefit in considering metrics that describe the artefact itself rather than the relation between two artefacts.

Two metrics that each measure an attribute of a given artefact have been chosen. These are the Lines of Code (LoC) metric and the Number of Static Elements within an artefact. As described above, these metrics differ in character in comparison to the name similarity and cohesion between objects measurement. These metrics do not measure the relation between two artefacts. They rather describe the artefact itself. Considering this, the research objective includes an examination also whether artefact metrics are able to guide the clustering search process. The next four sections describe the chosen metrics in detail.

Cohesion Between Objects (CBO)

The measurement of cohesion and coupling is of central importance in the field of software clustering (Jiang et al., 2007; Mitchell, 2002; Seng et al., 2005). The application of cohesion and coupling to clustering software systems is also meaningful for this project. This research project focuses on the decomposition of object oriented software systems. Hence, the applied metric particularly considers the examination of the influencing factors of cohesion and coupling in object oriented systems. The result is an implemented metric class *CohesionBetweenObjects* that measures the cohesion between two artefacts on the basis of the given source code and incorporates object oriented aspects. If a dependency from artefact A1 to artefact A2 exists (e.g. A1 calls/implements/inherits A2), then the *CohesionBetweenObjects* is calculated counting the direct usage of the class name occurrence, the method calls and the direct field accesses from A1 to elements of A2. The sum of the class, method and field occurrences is divided by the number of elements (class,

method and field) of artefact A2. Every occurrence of the class name, method and field has the same weight. No special weight is included for the interface or inheritance relationship as the depending artefact should use a high amount of the code elements of the interface or of the abstract artefact. Considering this, the implements and inheritance relationship is rewarded. For the clustering on package level, the relevant measurements are compressed to one measurement for every package dependency and divided by the number of dependencies between these packages.

Correlation of Names (CoN)

It has been shown in Anquetil and Lethbridge (1999b) that the similarity of artefact names can produce meaningful results in terms of clustering. This depends mainly on the patency of the system naming conventions. In their study, Anquetil and Lethbridge (1999b) constrained the name similarity measurement to the exposure of file names. For this project, with its particular focus on java projects, the inclusion of class and package names is meaningful. The package naming is relevant, because it can be driven by the implementation architecture of the system. The similarity is calculated by identifying the length of the longest identical substring for the names of two artefacts. A ratio value is created by dividing this value through the length of the longer string. The *CorrelationOfNames* metric determines the similarity measurement of class and package names on class level with an equal weight. For the clustering on package level only the package name is considered.

Lines of Code (LoC)

The LoC metric is a low level measurement, which counts the number of lines of code within an artefact. As stated in section 2.4, high measurements of the LoC metric can be indicative of high complexity of an artefact. Within the research community the LoC metric is often criticised as being unable to identify complex artefacts that may not be long or to estimate the evolution of a software system (Fenton & Pfleeger, 1997; Rosenberg, 1997). However, the LoC metric is easy to understand, independent of the applied artefact level and

imposes low computational complexity. Additionally the LoC metric examines a different characteristic of the software system than the *CohesionBetweenObjects* and *Name Similarity* measurement. Considering this, the application of the LoC metric is appropriate in this project. The class *TotalLOCMetric* metric comprises the functionality for measuring the lines of codes within a given artefact. To enable the clustering on package applying the *TotalLOCMetric* the sum of every artefact is calculated.

Number of Static Elements

The class *StaticElementsMetric* comprises a metric that counts the occurrence of static elements within an artefact. The employment of static elements bypasses the object oriented encapsulation principle, which is discouraged as the static elements are visible and accessible system wide. This increases the coupling between artefacts and erodes the modularisation of the system. As a result, frequent occurrence of static elements is generally an indication of a poor software design. For the clustering on package level is the sum of every artefact calculated.

Based on the described requirements and the selected metrics a framework can be designed. Figure 4.4 illustrates the class design of the metric framework. This follows the strategy design pattern, introduced by Gamma, Helm, Johnson and Vlissides (1995). The *MetricEngine* class symbolises the context of the metric framework and administers and encapsulates the metric behaviour. The interface *MetricStrategy* defines the interface for all specific metrics. Additionally to the strategy design pattern the abstract class *AbstractMetricStrategy* is introduced as an implementor of the *MetricStrategy* interface to comprise the overlapping functionality of the metrics. These overlapping requirements are the defined weighting of a metric and of the selected *Java* project. They have to be accessed by the metric implementation to calculate the metric value for a concrete artefact. The metric implementations inherit from the abstract class *AbstractMetricStrategy*. The metric framework stores the measured value directly into the node or the relation of the analyzed *Jung* graph.

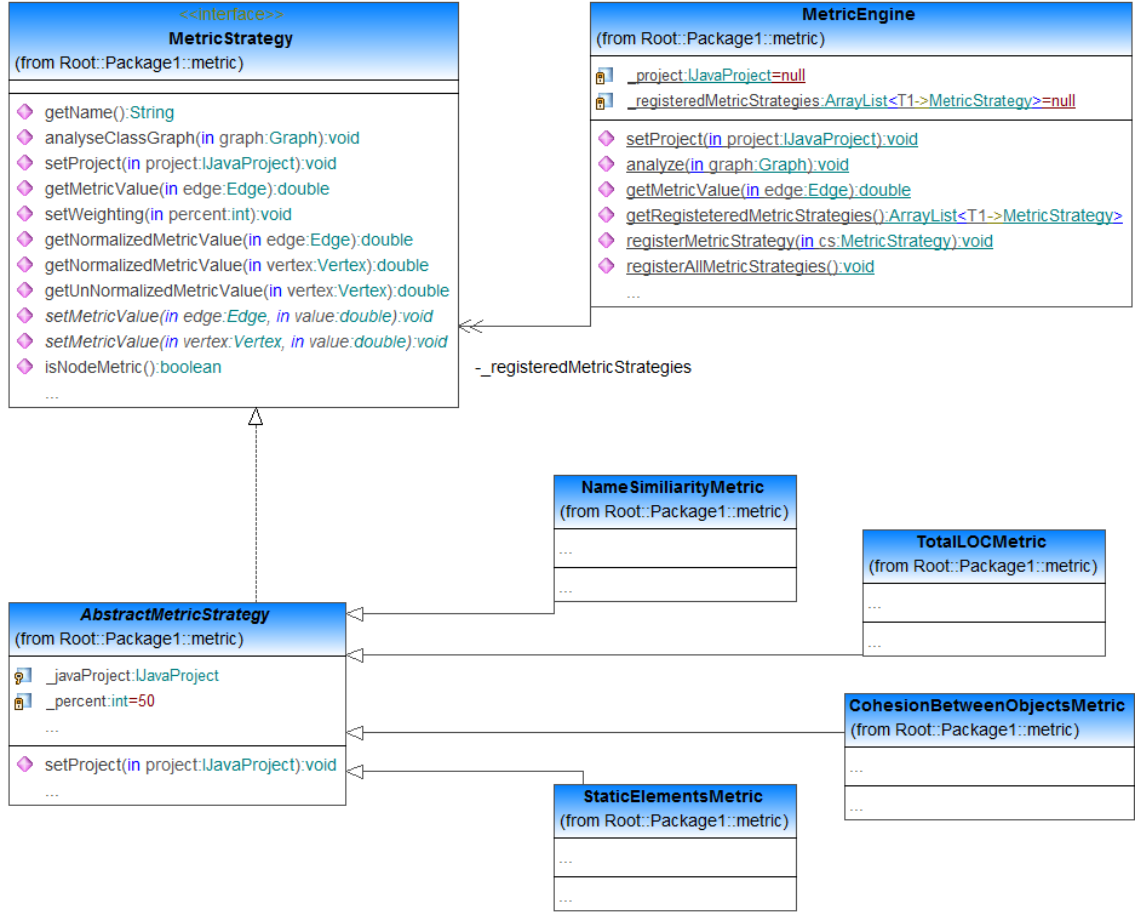


Figure 4.4 : Illustration of the design of the SBRE metric framework

The framework is able to analyze a *Jung* graph and to assign the containing nodes and relations with metric values. The framework is extensible, meaning that other metrics can also be integrated. For each metric a weighting can be set by the user to reflect its importance relative to other metrics. This weighting is set as a ratio value at the concrete *MetricStrategy* instance. The *getMetricValue(Edge)* combines all registered metric values and considers their weighting. The following formula describes the calculation of the normalized metric value where $S = f$ is a set of registered *MetricStrategy* instances and e (as instance of the type *Edge*) represents a dependency between two artefacts.

$$NormalisedMetricValue(e) = \frac{\sum_{i=1}^n Si.get\ Value(e) * Si.get\ Weight(e)}{n}$$

If a metric describes an artefact and not the relation between two artefacts, the measured value has to be stored in the vertex of the graph e.g. *TotalLocMetric* and

StaticElementsMetric. As the edge stores the connected vertices, the edge can gather the metric information from the destination vertex.

An additional objective of the research is to examine whether it is possible within an SBRE based cluster approach to identify code smells at a higher abstraction level. For the achievement of this objective it is not meaningful to cluster all of the artefacts of the software system, rather it is necessary to consider only artefacts that feature attributes which are indicators of code smells. For the realization of this requirement the user should be able to define a threshold. The exceeding of this threshold determines that an artefact should be considered during the clustering. In terms of the metrics selected for use here, it should be noted that the definition of a threshold for the *CohesionBetweenObjects* and *NameSimilarity* measures is not meaningful, as these metrics describe the *relation* between two artefacts and do not indicate a code smell. On the other hand, the *TotalLOCMetric* and the *StaticElementsMetric*, as node metrics, are suitable to derive an indication for a code smell. Thus, the user can define thresholds for these node metrics to reduce the set of clustered artefacts.

Based on this description of the design of the metric framework, the corresponding SBRE component is illustrated in Figure 4.5, which enables the user to configure the metric framework.

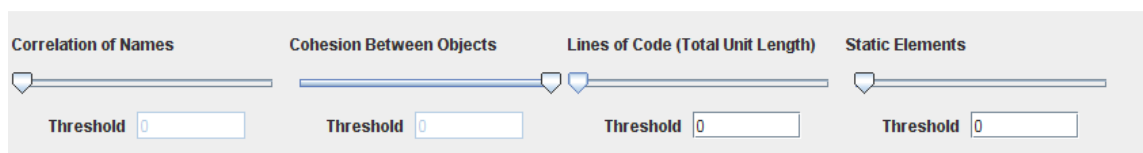


Figure 4.5 : Illustration of the metric configuration component

The sliders underneath the metric labelling enable the user to manipulate the weighting of the corresponding metrics. Each slider determines the weight in percent from zero to one hundred of the corresponding metric. Respectively, the positioning of the slider at the right side includes the measured metric value at one hundred percent in the similarity measurement. Configuration at the left side excludes the metric from the similarity measurement. Below the slider the input fields are arranged to indicate the threshold values which serve to limit the number of clustered artefacts. A

threshold of zero defines the threshold as not active. A value larger than zero determines that every artefact with a value lower than the corresponding threshold value is excluded from the cluster analysis. As the *Correlation of Names* and *Cohesion Between Objects* metrics are not suitable for the application of a threshold, the fields for these metrics are not active and no threshold can be determined.

Based on the measurements of the artefacts and of the relationships between the artefacts in correspondence with the configuration of the metric framework, the cluster search can be executed. For this the cluster algorithms have to be designed and implemented.

4.4 Design and Implementation of the Cluster Algorithms

The measurements described above represent the decision basis for the clustering process. As described in section 2.2.2, Talbi (2009) suggested a framework for the design and implementation of metaheuristics into the three phases of solution representation, design of the objective function and constraint handling.

4.4.1 Solution Representation

It has been demonstrated in section 2.3.2 that the software clustering problem is a graph partitioning problem. It has been illustrated in section 4.2 that the software system is available as a *Jung* graph instance. A requirement for a search based clustering framework is the easy manipulation of the graph and the storage of multiple variations of the graphs. Additionally, it should be relatively straightforward to compare and evaluate a graph. The *Jung* graph framework does not fulfil these requirements. The graph stores information which is irrelevant for the clustering process and which hinders their use in the area of search based software engineering. Additionally, the interface of the *Jung* graph framework is not optimized for application within a search based environment. Considering these reasons, a different lightweight representation is designed.

Within an implementation perspective the artefacts and dependencies between these artefacts has to be visualized. In object oriented systems three different kinds of dependencies exist: the “uses”, the “implements” and the “extends” dependency.

Each is associated with two classes and has a direction to indicate which is the depending class. Given this, a software system can be modeled as a directed graph $G = (V; E)$. The set of nodes represents the artefacts of the software system and the set of edges $E \subseteq (V \times V)$ represents the dependencies between these artefacts. As stated in section 2.3 the goal of software clustering is the partitioning of a software system into meaningful clusters. A cluster comprises a set of artefacts. One artefact can only be a member of one cluster, and every cluster in the system contains at least one artefact.

Based on this, a class model can be derived to represent the clustering of a software system. Figure 4.6 illustrates the corresponding model with the classes of the *SBRE* representation.

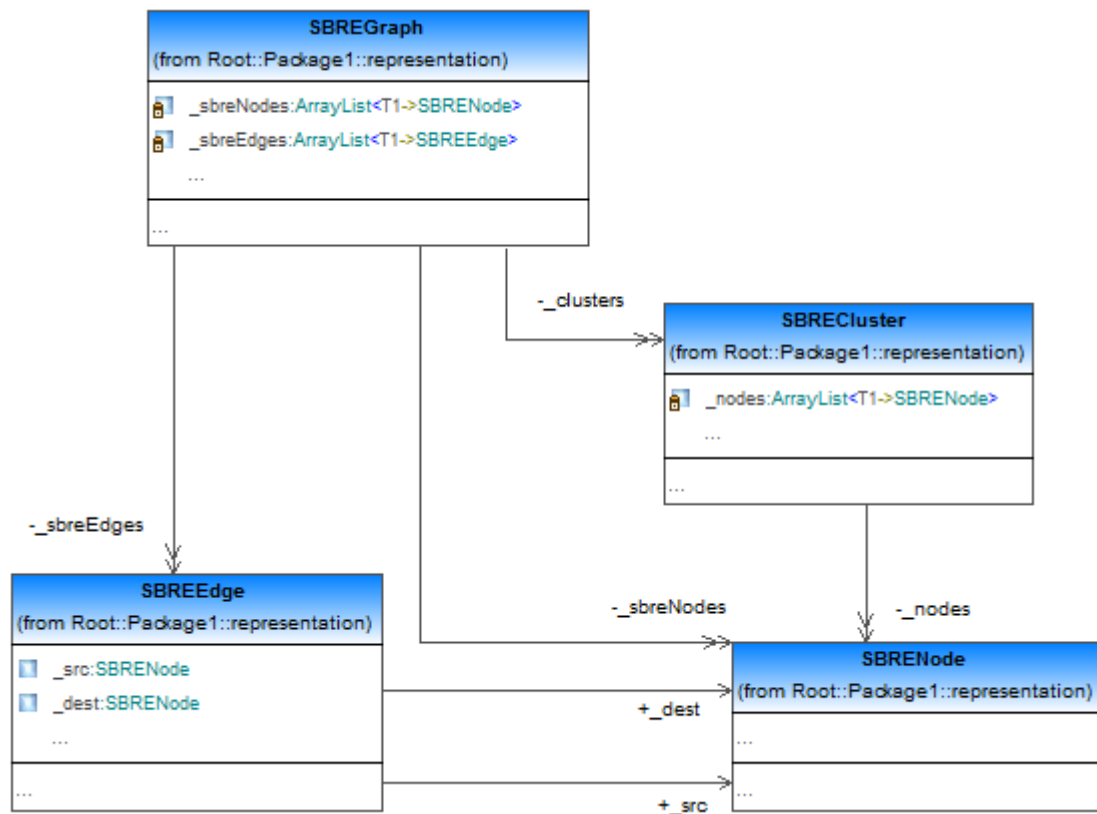


Figure 4.6 : Class diagram of the SBRE representation

An instance of the *SBRENode* represents an artefact of the software system. The *SBREEdge* embodies the dependencies between the artefacts. The field `_value` represents the combined metric value of the applied metrics including the weight adjustment of the user.

The *SBREEdge* class associates two instances of the class *SBRENode*. This is illustrated by the fields *_src* and *_dest* and fulfils the requirement to represent the direction of dependencies. The class *SBREGraph* relates to a set of *SBRECluster* instances. An instance of the class *SBRECluster* represents a cluster. The class *SBRECluster* comprises a set of *SBRENode* instances as a representation of the artefacts that are contained in the cluster.

An instance of the illustrated model is not only the representation of the artefacts, dependencies and similarities between these artefacts, it also represents a solution within the search space if all nodes of the graph are assigned to clusters.

4.4.2 Fitness Function

The fitness function evaluates the quality of a solution and enables the comparison of candidate solutions. As this research focuses on the application of multiple metrics and incorporates functionality that enables the user to adjust the weighting of these metrics, the fitness function includes these user adjustments and the selection of metrics to evaluate solution quality.

The fitness function for this project promotes high cohesion within the clusters and penalize high coupling between the clusters. This rewards solutions which feature a high cohesion of the configured metric weighting and finally reflects the assumptions of the user. Compared to other approaches (Mitchell, 2002; Seng et al., 2005) that also promote high cohesion and low coupling, in this case the numerical base does not only reflect the dependency binding between the connected artefacts, it also reflects the measured metric values and the configured weighing of the user. To assess this, the measurement of *ClusterCohesion* is introduced. The *ClusterCohesion* is inspired by the *ClusterFactor* introduced in Mitchell (2002), but considers in comparison to Mitchell (2002) the configured metric weighting.

The cluster quality for a cluster *c* which includes a set of artefacts $A = \{a_1, a_2, \dots, a_n\}$ is calculated as follows:

$$ClusterCohesion(c) = \frac{\sum_{i=1}^{|A|} ai.getValue()}{|A|}$$

The *ClusterCohesion* represents the cohesion measurement for one cluster of a solution. The cluster quality has to be calculated for all existing clusters. This sum is then divided by the number of existing clusters. So we define $C = \{c1, c2, c3, ..., cn\}$ as the set of clusters within a solution representation.

$$AverageClusterCohesion(G) = \frac{\sum_{i=1}^{|C|} Ci.getClusterQuality()}{|C|}$$

As described previously the coupling of clusters is also of relevance for the construction of the fitness function. To examine the coupling between two clusters the edges are of relevance, as these act as a link between two clusters. This means that the source artefact representation is dedicated to another artefact than the destination artefact representation. Figure 4.7 illustrates two clusters with a linking edge between them.

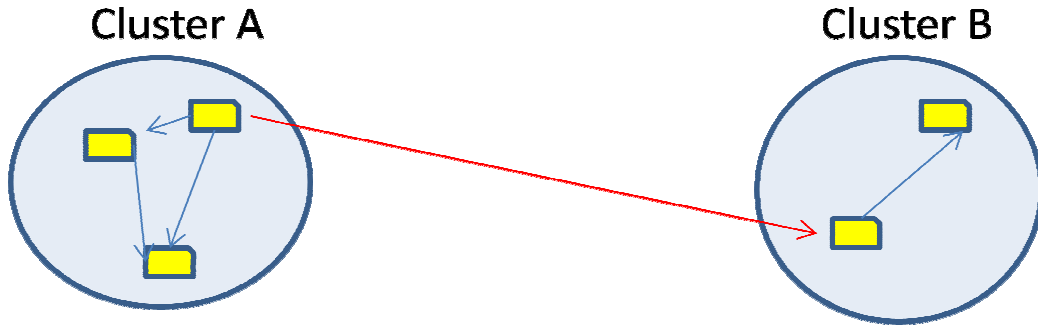


Figure 4.7 : Illustration of the coupling between clusters

As described previously, a software system within this work is represented as a graph $G=(V;E)$ with $E \subseteq (V \times V)$. The set of edges which are relevant for the coupling analyses as:

$$S = \forall e \in E: e.getDest().getCluster() \neq e.getSource().getCluster()$$

Based on this set the *ClusterCoupling* of the graph G can be calculated.

$$ClusterCoupling(G) = \frac{\sum_{i=1}^{|S|} Si.getNormalisedMetricValue()}{|S|}$$

Based on the *AverageClusterCohesion* and the *ClusterCoupling* the quality of the solution can be calculated. The following formula supports high cohesion and penalizes high coupling and delivers the basic fitness function for the evaluation of solutions within this research.

$$SolutionQuality(G) = \frac{AverageClusterCohesion(G)}{ClusterCoupling(G)}$$

Note that if the *AverageClusterCohesion(G)* or the *ClusterCoupling(G)* return zero as a result, the calculation cannot be determined and it is an invalid solution.

Additionally, a clustering should ideally result in a homogeneous decomposition of the software system. As illustrated in Figure 4.8 a consistent artefact size can reduce the complexity of the software system. This principle can also be assigned to the size of clusters within a software system. It motivates that very small or very big clusters should be penalized, depending on the total number of clusters in the software system.

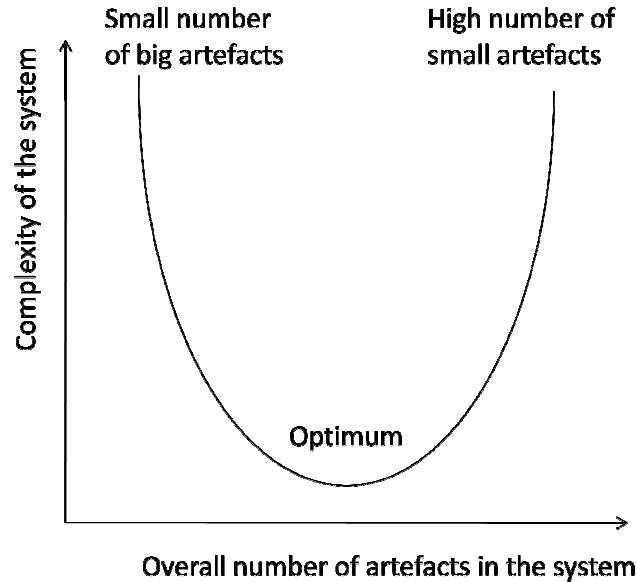


Figure 4.8 : Complexity of a system in relation to the granularity: adapted from Baas (2003)

This research adopts the principle represented in Figure 4.8 and derives the penalization for excessively small or large clusters from the characteristic of a parabola. First the angular point for the optimal cluster size has to be defined. For this research project the optimal cluster size is defined as 10 percent of the entire artefact count. Hence, approximately ten clusters are included in the system. It is fair to say that this approach is naïve and does not consider that with larger systems it would be necessary to have more clusters in the system. But to illustrate if a parabola function is adequate, this assumption is appropriate at this stage.

The application of a parabola as a penalizing instrument has two disadvantages. First, every deviation from the determined optimum would be penalized. However, depending on the system or the classification in the system architecture, the optimal size of a cluster can vary. This problem can be attenuated with a shallow ascent around the angular point. Additionally, as a second potential drawback, parabolic functions are isosceles around the angular point. Thus a given deviation in a positive or negative way based on the optimum causes the same function result. This can cause problems when the function results should differ regarding the direction of the infraction. However, this is not relevant in the application of a penalizing function in the area of software clustering, because every solution with excessively small clusters should also incur the same penalty as a solution with too many oversized clusters. Hence the application of a parabola function provides appropriate penalization in the evaluation of software cluster solutions.

As determined previously the anchor point is set at 10 percent of the entire artefact count. The following formula is derived to penalize the system function. The solution $s \in S$ represents any solution of the search space. The set $C = \{c_1, c_2, \dots, c_n\}$ defines the set of clusters of a given solution s . $x \in \mathbb{R}$ is defined as the ratio of artefacts within a certain cluster $c \in C$ depending on the total number of artefacts within the system:

$$x = c.getArtefactRatio()$$

$$f(x) = (x - 0.1)^2$$

Transformed to:

$$f(x) = x^2 - 0.2x + 0.01$$

Additionally the function has to be normalized. For this research project it is assumed that a solution, which exhibits more than fifty percent of the entire system artefacts, is not feasible and should generate a penalty that defines the solution as completely infeasible. The penalty is given as ratio value and therefore, a penalty of 1.00 signifies a solution as completely infeasible. As a result of this derivation the following formula is defined to penalize a cluster:

$$f(x) = 6x^2 - 1.2x + 0.06$$

This formula represents the penalty for one cluster. It is understandable, that every cluster of a solution has to be penalized and be added up to a total solution penalty. It is accepted that solutions with a higher number of clusters are more strongly penalized under this approach.

$$\delta(s) = \sum_{i=1}^{|C|} f(ci.getArtefactRatio())$$

In conclusion, the surjective function $\delta: S \rightarrow \mathbb{R}$ supports the identification of inhomogeneous solutions. To derive an indication of a final solution quality the fitness function and the penalizing function have to be combined.

The previously introduced *SolutionQuality (SQ)* fitness function is a surjective function $f: S \rightarrow \mathbb{R}$, which transforms a solution from the solution space S into the numerical relational system \mathbb{R} . Additionally a penalizing function $\delta: S \rightarrow \mathbb{R}$ has been defined which evaluates and penalizes the lack of homogeneity of a given solution. The combination of the fitness function and the penalizing function $f'(s) = f(s) * (1 - \delta(s))$ determines the penalized solution quality for any solution of the solution space $s \in S$. Finally, the combined fitness function $f'(s)$ enables the comparison of individual solutions considering the aspect of homogeneity. The fitness function $f'(s)$ is offered as the method `getSolutionQuality()` as a part of the *SBREGraph* interface.

4.4.3 User Constraints

An important part of this research is the evaluation, which comprises approaches that can improve the effectiveness of the search process. As stated in section 4.1, the approach adopted here utilises an interactive clustering process, which allows the user to include data in the process to guide the search into more promising areas of the search space. The design strategy is to include this additional user data as constraints on the clustering process. The requirements are therefore to administer cluster constraints, to verify, if a solution candidate adheres to the configured constraints and to generate initial solutions that align with the configured constraints. To fulfil these requirements the class *ConstraintEngine* is designed, which comprises a set of *ClusterConstraint* instances. Figure 4.9 depicts the class diagram for the constraint component:

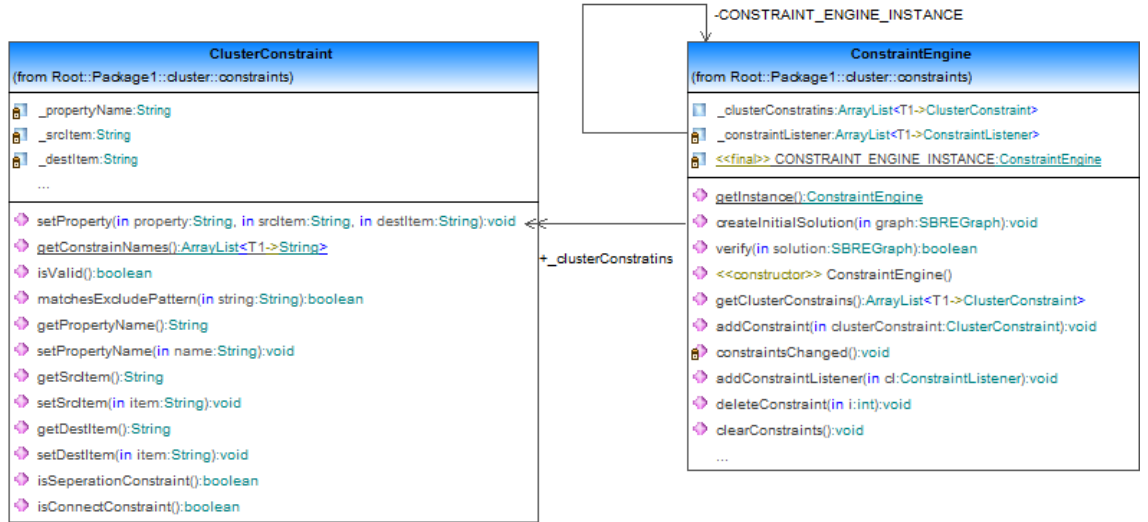


Figure 4.9 : Class diagram of the SBRE cluster constraint framework

Besides the administration of constraints with the methods *addConstraint()*, *deleteConstraint()* and *clearConstraints()*, the degree to which a solution aligns with the configured constraints can be determined using the method *verify(SBREGraph solution)*. Solutions that do not align with the configured constraints are rejected. A *ClusterConstraint* instance accepts three parameters, which are the two constraint artefacts and the type of the constraint.

Three constraints have been identified and implemented during this research to force the search into different areas of the search space. The *Combine Constraint* accepts

only solutions that combine two determined artefacts in the same cluster. The converse is the *Separate Constraint*, which only accepts solutions that place the two given artefacts in separate clusters. The *Exclude Constraint* excludes artefacts from the cluster process. This latter functionality is useful to reduce the search space for the identification of “code smells” or to exclude irrelevant artefacts from candidate solutions.

4.4.4 Metaheuristics

The core of this research is the process of creating a solution candidate within the search space. This task is executed by the search algorithm. As previously described, this work focuses on metaheuristics to solve the software clustering problem and especially on the application of tabu search. To apply tabu search, a start solution is necessary. To create this initial solution, an algorithm is needed which does not assume an initial solution. As described in section 2.2.3, the greedy algorithm is a metaheuristic that is able to create a solution from scratch. Hence, it is meaningful for this research project to implement a greedy algorithm and tabu search algorithm individually. However, before the design of these algorithms can be pursued, a framework has to be designed, within which these metaheuristic algorithms can be applied.

Design and Implementation of the SBRE Cluster Framework

The responsibility of the cluster framework is the management of the cluster algorithms and the triggering of the clustering process. Figure 4.10 illustrates the design of the cluster framework.

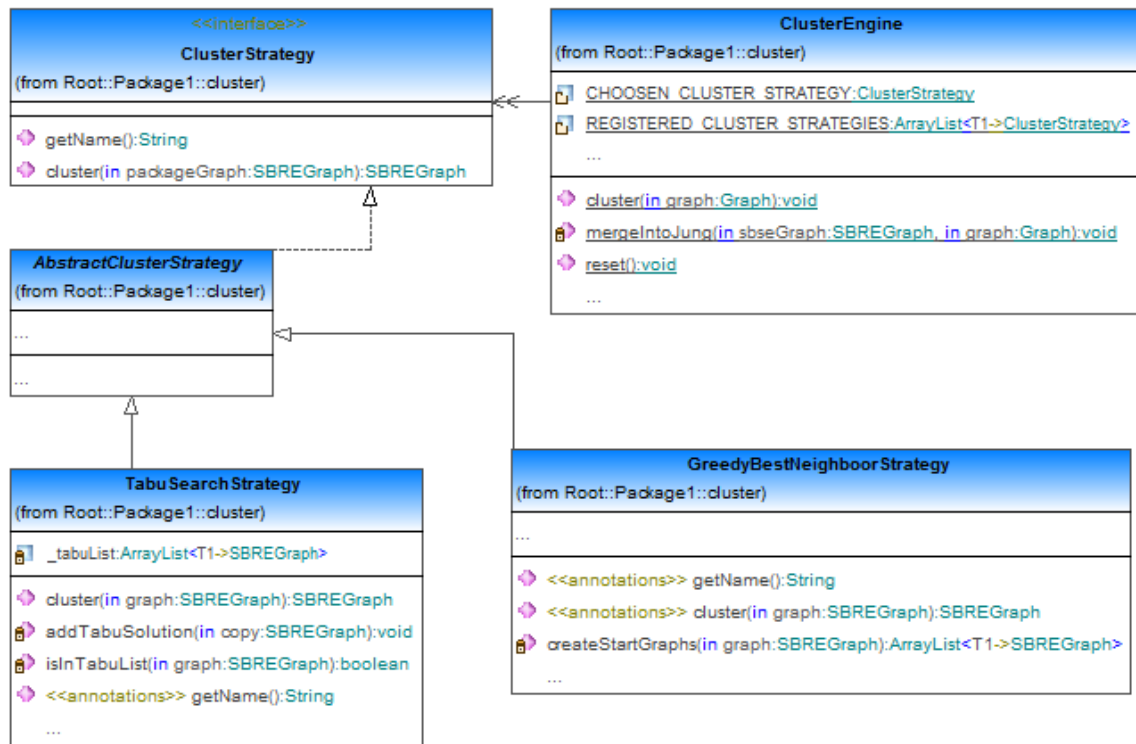


Figure 4.10 : Class diagram of the SBRE cluster framework

The class *ClusterEngine* represents the communication interface for the system. The *ClusterEngine* manages a set of registered *ClusterStrategy* instances. One *ClusterStrategy* instance is the currently active cluster strategy. The most significant functionality of the *ClusterEngine* class is the *cluster()* method. Within the *cluster* method the *Jung* graph instance is transformed to a *SBREGraph* instance. The *SBSEGraph* instance is handed to the currently selected cluster algorithm. The cluster algorithms are implemented following the design strategy pattern concept by Gamma, Helm, Johnson and Vlissides (1995) with the additional inserted abstract class *AbstractClusterStrategy*. The abstract class *AbstractClusterStrategy* pools overlapping functionality to enable code reuse. Specific cluster algorithms inherit from *AbstractClusterStrategy*. As previously described the SBRE framework offers the two *ClusterStrategy* implementations *GreedyBestNeighbour algorithm* and *TabuSearchClusterStrategy*, which represent the implementation of the greedy and tabu search clustering respectively. The design and implementation of these algorithms is described in the following sections.

Design and Implementation of the Greedy Algorithm

Within this research a Greedy algorithm has been designed. The functionality of this greedy algorithm is encapsulated in the class *GreedyBestNeighbour*. The objective of this search process is the assignment of artefacts into an unknown number of clusters. As described, greedy algorithms usually start from scratch.

Hence, the *GreedyBestNeighbour* algorithm can start from scratch or build up from a preconfigured incomplete solution. This behaviour has been chosen to incorporate user data into the clustering process. The received initial solution includes the executed user constraints as initial clusters and assigned artefacts. Based on this, the algorithm then classifies the unassigned artefacts to clusters. If an artefact should be assigned, it can be included in one of the existing clusters or it can be assigned into a new cluster. The objective of this process is to isolate depending artefacts with high similarity. Given this, it is not the aim of this research to combine artefacts in clusters that exhibit high similarity, but in clusters which are not depending on each other. Therefore the dependency between two artefacts is of central importance for the assignment into one cluster. This requirement also reduces the search space, as two artefacts can only be members of the same cluster when a path exists between these two artefacts.

The algorithm starts with selecting the next unassigned artefact from the graph representation. Based on this current artefact all incoming and outgoing edges to or from the artefact are identified. Based on the metric value of these edges the best suitable neighbour is identified. The metric value represents the combination of all registered metrics including the weight adjustment. If the best neighbour is already a member of an existing cluster the current artefact is also assigned to this cluster. Figure 4.11 illustrates the assignment of one unassigned artefact into an existing cluster.

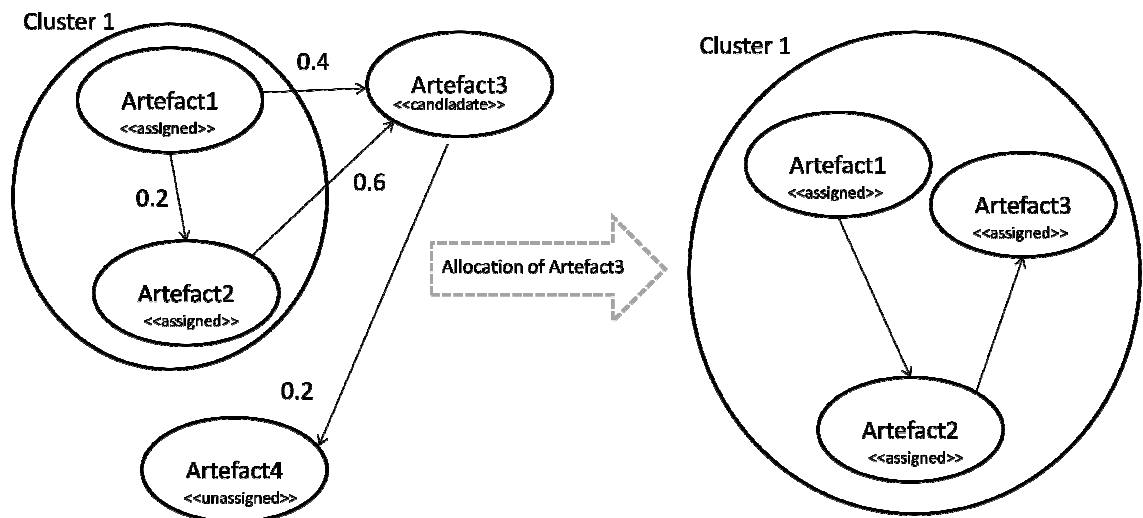


Figure 4.11 : Allocation into existing clusters of the SBRE greedy algorithm

If the best neighbour is not a member of an existing cluster, a new cluster is created and the current artefact is the first member of this new cluster, as illustrated in Figure 4.12. The greedy algorithm considers only the strongest dependency for the decision process. It could be possible that two weak dependencies exist, which lead into one cluster, which then would certainly indicate that the candidate could be a member of that cluster. This circumstance is not considered by the algorithm at this stage. Figure 4.12 illustrates that the artefact is assigned into a new cluster, even though there exist two connections to artefacts from *Cluster1* exist.

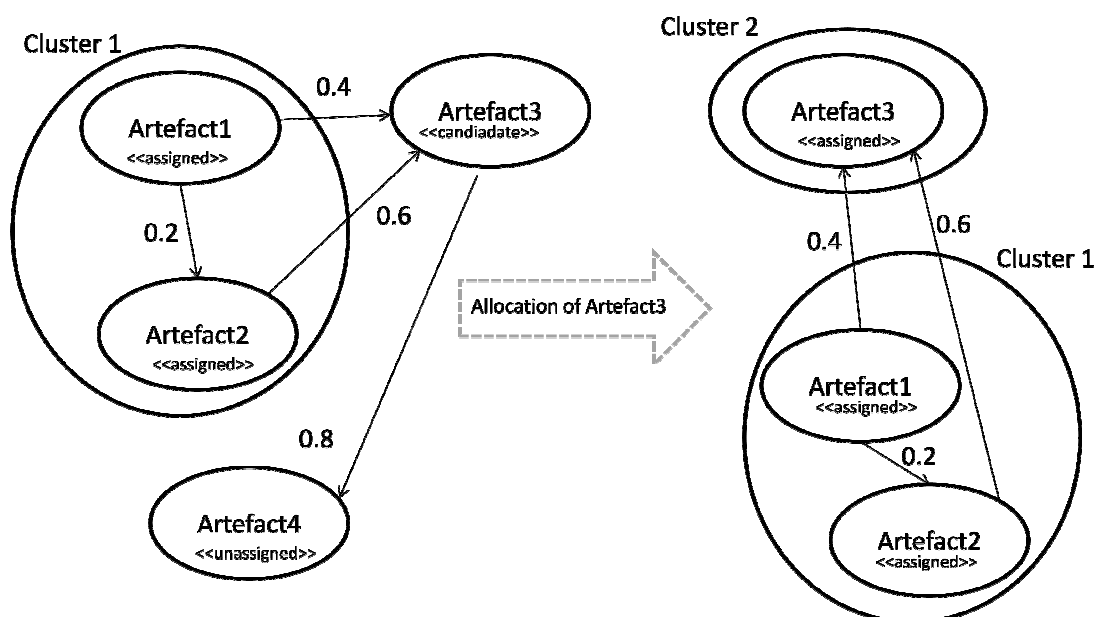


Figure 4.12 : Assignment of artefact into a new cluster of the SBRE greedy algorithm

To overcome the problem of stalling in local minima, the algorithms start a separate search for every possible unassigned artefact. The best solution is evaluated using the fitness function introduced in section 4.4.2. The designed greedy algorithm is offered as a separate strategy in the SBRE tool, but also delivers the starting point for the design and implementation of the tabu search algorithm.

Algorithm 4.1 illustrates the pseudo code for the cluster method of the class *GreedyBestNeighbour* algorithm. The illustrated *cluster* method is called for every artefact of the graph as a start node. After the cluster method delivers a result with every artefact as a start solution, the best solution is selected using the fitness function.

```

cluster(SBSEGraph initalSolution, SBRENode startNode)

Let G = initalSolution.copy()
repeat
    Let N = G.getNodesWithoutCluster()
    Let C = G.getCluster()
    Let node = startNode
    Let bestMetricVaue = 0;
    Let bestCluster = null;

    if (startNode == null) then
        node = N.getFirst()
        startNode = null
    end

    foreach (c ∈ C) do
        foreach (n ∈ c.getNodes()) do

            foreach (e ∈ c.getEdges()) do

                if (e.getMetricValue >= bestMetricValue) then

                    bestCluster = c
                    bestMetricValue = e.getMetricValue()
                end
            end
        end
    end

    if (bestCluster != null) then
        node.setCluster(bestCluster)
    else
        cluster = createCluster()
        node.setCluster(cluster)
    end

until "all nodes without cluster are assigned into clusters"

return G;
    
```

Algorithm 4.1 : PseudoCode of the *GreedyBestNeighbour* cluster method

Design and Implementation of the Tabu Search Algorithm

One aim of this research is to assess the applicability of tabu search in the field of software clustering. To address this aim the class *TabuSearchStrategy* is designed within this research project.

Tabu search algorithms require an initial solution as a basis for further solution improvement. To create an initial solution the *GreedyBestNeighbour* algorithm, which has been described in the previous section, is applied within the *TabuSearchStrategy* clustering process. By default, the best identified solution from the class *GreedyBestNeighbour* is used as the initial solution for the *TabuSearchStrategy*. Additionally, the previously clustered graph can be used as the initial graph for the

tabu search clustering. This graph can be created from the *GreedyBestNeighbour* algorithm or from the *TabuSearchStrategy* itself. This enables the user to change the metric configuration and constraints at any time in the search to guide the search into different areas of the search space.

Based on this start solution, the search can begin to identify improved solutions. While the *GreedyBestNeighbour* algorithm follows a relatively naive approach of classifying the system, the *TabuSearchStrategy* on the contrary explores more solution candidates of the search space and as a result can hopefully seek better results than the greedy algorithm. To simplify the illustration of the main components and method of operation of the *TabuSearchStrategy* algorithm, the interface of the *TabuSearchStrategy* is portrayed in Figure 4.13.

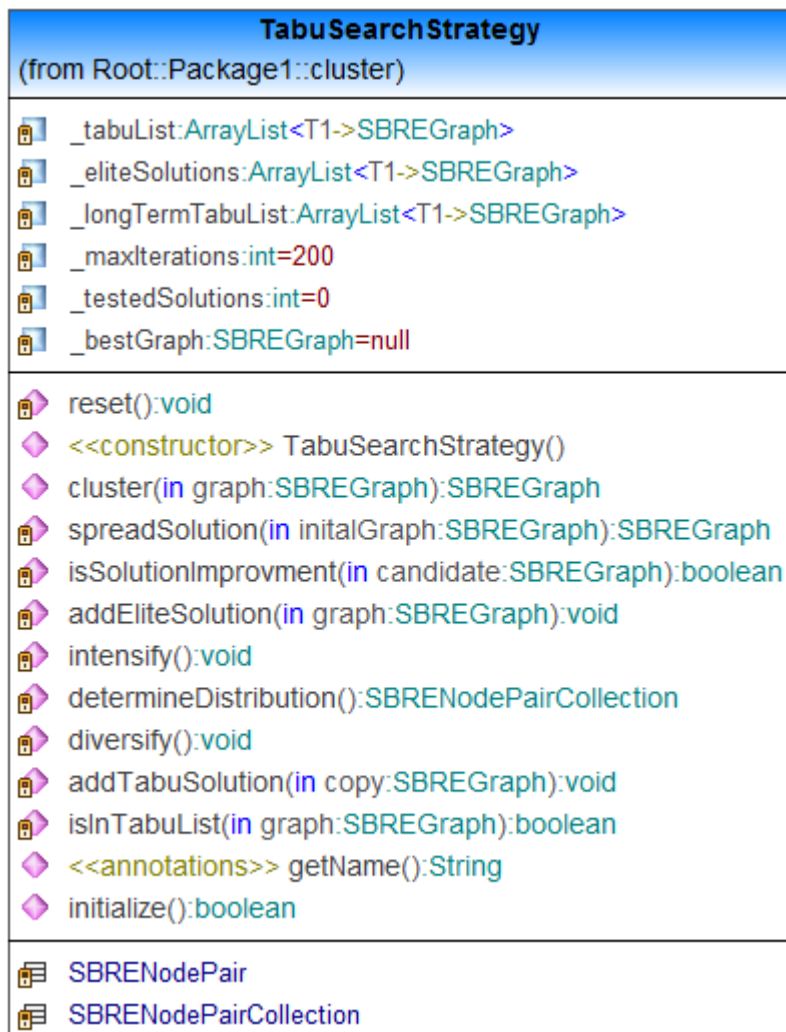


Figure 4.13 : Class diagram of the *TabuSearchStrategy* algorithm

The core algorithm is implemented in the method *spreadSolution(SBREGraph initialGraph)*. Because of its importance in this research, this method is described in more detail. The *spreadSolution* method receives an instance of the *SBREGraph* class as an argument. The *SBREGraph* instance represents a solution of the search space, including a set of clusters with the assigned *SBRENode* instances. The second parameter *nodeCandidatesForDistribution* contains a set of *SBRENode* instances as representation of artefacts. These *SBRENode* instances of artefacts are members of the *SBREGraph* solution. As a default, the *nodeCandidatesForDistribution* set contains every *SBRENode* instance of the solution. It can also contain a subset of *SBRENode* instances; this is of interest if the search should be intensified or diversified with a special focus on this subset of *SBRENode* instances.

The cluster information of the received nodes is reset by the algorithm. To prevent the same solution being identified within the next generation, the algorithm refuses to classify an artefact into the same cluster. This anticipates unnecessary comparisons with the members of the tabu list, because this branch is already examined with the current search iteration. The solutions of the new solution set represent the initial solutions for the next iteration of the search.

An example should help to illustrate the method of operation of the tabu search algorithm. The initial solution S_i consists of two clusters $C = \{c1, c2\}$. Every cluster contains a set of artefacts $c1 = \{a1, a2, a3\}$ and $c2 = \{a4\}$. To simplify the illustration a solution is given as a set with an amount of subsets. Every subset represents a cluster with the associated artefacts. Regarding this the representation for the initial solution is $S_i = \{\{a1, a2, a3\}, \{a4\}\}$. The set of artefacts which should be distributed is defined as the complete set of artefacts within the solution $A = \{a1, a2, a3, a4\}$. Figure 4.14 visualizes the first search generation of this example. The example illustrates, based on the initial solution, the creation of four new solutions within the first generation. Every created solution represents a solution candidate. The solution candidate is evaluated by the developed fitness function. The best solution out of this set is chosen using the fitness function. This “best” solution represents the base for the next search iteration.

In this example it is assumed that $\{\{a1, a2\}, \{a3, a4\}\}$ represent the best solution, the other solutions are discarded.

Figure 4.14 drafts also the second generation of the search with $\{\{a1, a2\}, \{a3, a4\}\}$ as the initial solution. As illustrated the nodes are only assigned into the existing cluster. It would certainly also be possible to include every assigned node into a new cluster.

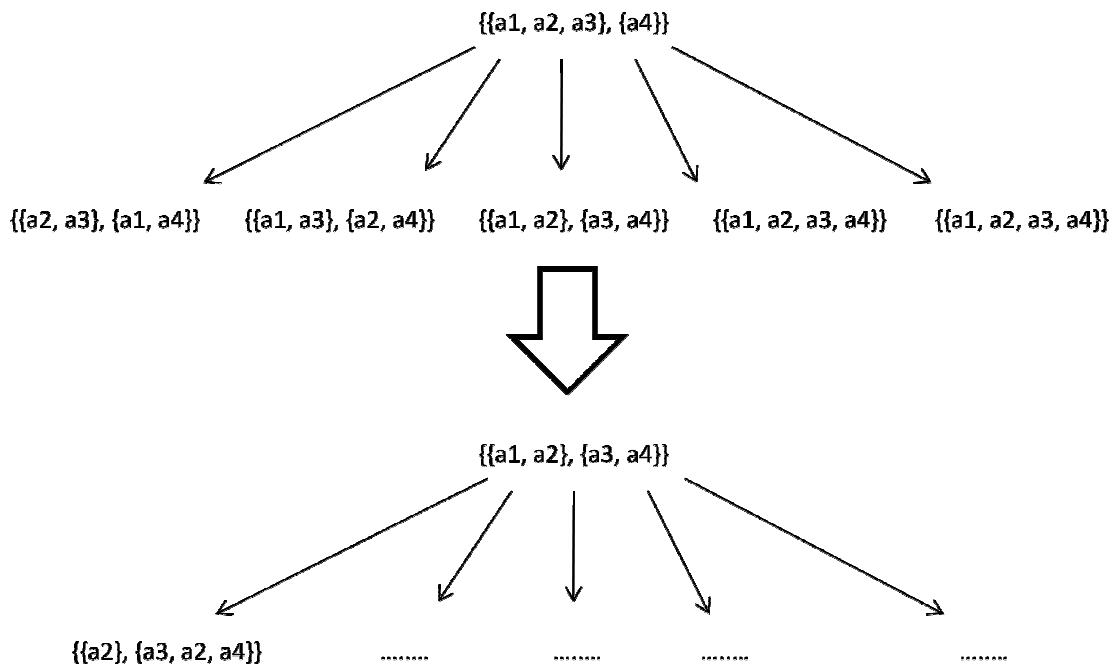


Figure 4.14 : Example of the clustering process of the TabuSearchStrategy algorithm

Evaluated solutions are stored in the tabu list and during the creation of a new solution candidate the list is examined to determine if the solution branch has already been investigated. If a solution is stored in the tabu list, the solution branch is not further examined. This prevents the algorithm from exploring previously investigated branches of the search space.

The maximum number of stored entries of the tabu list can be determined by the user. A long tabu list will prevent the occurrence of cycles and the stalling of the algorithm within an infinite loop. However a longer list will also decrease the performance of the algorithm.

In addition to the length of the tabu list, the number of maximum created solution candidates can be determined. This value can be set by the user to manipulate the

depth of the search. A higher number of tested solutions increases the chance of finding better solutions within the search space, but on the other side it increases the required computational time of the search.

Algorithm 4.2 illustrates the pseudo code of the `spreadSolution` code, the main algorithm of the implemented tabu search metaheuristic.

```

spreadSolution(SBSEGraph initalSolution)

    Let B the current best solution
    Let N be the set of all artefacts of the initalSolution
    Let C be the set of all clusters of the initalSolution
    Let S be an empty set of solutions/SBREGraphs

    if (testedCandidates() > maxTestedCandidates()) then
        return B
    end

    foreach n ∈ N do
        foreach c ∈ C do

            Let I = initalSolution.copy()
            Let c2 = I.getSBRECluster(c)
            Let n2 = I.getSBRENode(n)

            n2.setCluster(c2)

            increaseTestedCandidates()

            //check constraints and tabu list
            if (verifySolution(I)) then
                S.add(I)
                if (isImprovingSolution()) then
                    B = I
                end
            end

            if (isTimeForIntensification()) then
                intensify()
            end

            if (isTimeForDiversify()) then
                diversify()
            end

        end
    end

    spreadSolution(S.getBestSolution())
end
    
```

Algorithm 4.2 : PseudoCode of the *TabuSearchStrategy* `spreadSolution` method

The algorithm illustrates that tabu search algorithms rely strongly on the creation, evaluation and rejection of solutions. This motivates the importance of a lightweight representation and an efficient comparison and duplication mechanism for solutions.

The *TabuSearchAlgorithm* also implements intensification and diversification strategies to overcome local optima and to approach the global optimum. The intensification and diversification are triggered by an idle value, which counts the number of non-improving solution candidates. The identification of an improving solution or the triggering of the diversification or intensification process resets the corresponding idle parameter. Both methods are controlled by their own idle values. Correspondingly, the search can be focused more on intensification or diversification. The number of idle iterations can be configured by the user.

The *TabuSearchAlgorithm* algorithm stores the elite solutions of the search in a separate list. For the proof of concept of this research project the ten best solutions are stored in the elite solution list. This simple approach is sufficient to examine the method of operation of tabu search within the field of software clustering. Certainly a more intelligent or flexible approach would also be possible, which would store the elite solutions depending on user input or input size parameters. From these elite solutions a new solution is derived, which exhibits the highest similarities of artefact combinations in one cluster. For example, if artefact A1 was the most combined artefact with A2, A1 and A2 are combined in one cluster within this new solution. This solution represents the initial solution for the next iteration of the search. Hence, the created solution is handed to the *spreadSolution()* method as an initial solution during the next search iteration. Based on this, the search will be intensified in this region of the search space.

If the algorithm cannot improve in the current area of the search space, the diversification strategy can be applied to guide the search into a different area of the search space. The implemented diversification method within this research is based on the homogeneity of the cluster landscape and combines small clusters and separates bigger clusters. The diversification method separates every cluster of the current best solution into two clusters, which has more than five artefacts. Sets of two clusters with

fewer than five artefacts are combined into one cluster and the remaining empty cluster is discarded. The selection of a fixed separation level is certainly naïve, but again it is not a central component of the research to evaluate the optimal diversification method in the area of software clustering. As such, a naïve diversification approach is sufficient within this research project.

As illustrated during the previous paragraph the *TabuSearchAlgorithm* is influenced by the “Idle Diversification”, the “Idle Intensification”, the “Maximum Tested Solutions” and the “Length of the Tabu List” parameters. These parameters can be configured by the user. Before every search, which applies the *TabuSearchAlgorithm* a configuration window is displayed, which enables the tuning of these parameters. Figure 4.15 illustrates the configuration window for the *TabuSearchAlgorithm*.

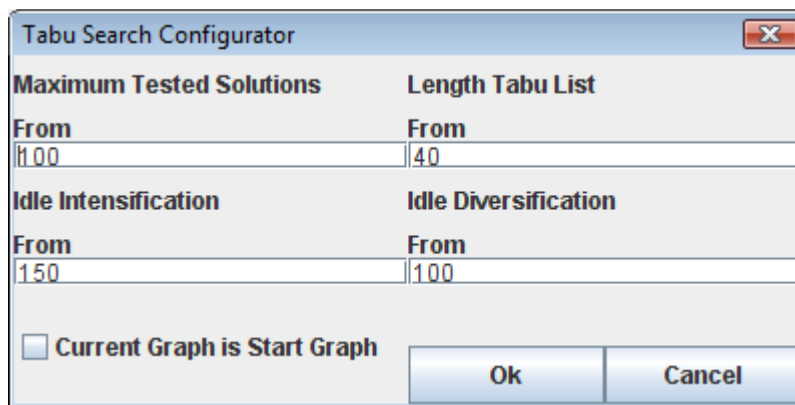


Figure 4.15 : Example of the *TabuSearchStrategy* configuration component

As stated, the *TabuSearchAlgorithm* can use any clustered *SBREGraph* instance as the initial start solution. By default, the best solution from *GreedyAlgorithmBestNeighbor* algorithm is used. However, it is also possible that a previously clustered graph can be the basis for further clustering. This allows the user to manipulate a graph with constraints or to change the metric configuration and to retrigger the clustering with this graph. To facilitate this behaviour the checkbox “*Current Graph is Start Graph*” has to be selected.

The next section provides an overview of the functionality of the SBRE system. As the purpose of the SBRE system is to enable the implementation and examination of the

research objective, this overview focuses on the necessary components and functionality to deliver against the research objective.

4.5 Functionality of the SBRE System

This section describes the functionality and application of the Search Based Rearchitecture Engineering (SBRE) component. The SBRE component acts as an enabler to examine the application of the greedy and tabu search metaheuristic in the area of software clustering and to examine the feasibility of a user-controlled clustering process. Regarding this, the SBRE component is a semi-automatic clustering component that allows the user to control the clustering process by adjusting metrics and by including user-constraints. The advantage of this approach is that the user can include domain knowledge into the clustering process and align the clustering with needed preferences. It is important to note that the metrics and constraints make no decisions for the user regarding the clustering. Rather, they determine the direction of the clustering. The following section gives an overview of the components and their functionality. Figure 4.16 displays a screenshot of the complete SBRE tool and names the relevant components. Based on this, the SBRE components are briefly described. The navigation, project selection and visualisation capabilities are plausible, but are not relevant for the examination of the research objective and as a result are not further illustrated here.

Cluster Component

Within this component the cluster process can be triggered by pressing the *Calculate* button. The applied metaheuristic can also be chosen within the *Heuristic Strategy* selection box. Additionally, the cluster level can be determined. The user can choose to apply the clustering on class or package level.

Constraint Component

The Constraint Component features a table that displays the active constraints. Every constraint consists of the type, which can be *connect-* or *disconnect elements*, the source and the destination artefact. A constraint can be created within the graph by a right click onto an artefact or manually by executing the *create* button underneath the

table. Additionally, a constraint can be edited within the table or deleted by selecting it and executing the *delete* button underneath the table.

Metric Configuration Component

The Metric Component enables the user to determine the weighting of the individual metrics by adjusting the sliders. Additionally, a threshold can be defined. Artefacts that have metric values, which fall below the defined threshold are excluded from the next cluster iteration. A threshold of zero signifies the threshold as not active. As stated previously, the threshold field for the *Correlation of Names* and *Cohesion Between Objects* metrics are not active.

Info Component

The info box displays cluster information. The number of artefacts, number of edges, the number of clusters and the solution quality is displayed. In addition, the information level can be changed by executing a left mouse click onto clusters of artefacts within the displayed graph.

Visualisation Component

The visualisation component displays the clustered graph with its artefacts and dependencies between the artefacts. The SBRE component features a horizontal, initial ordering of the clusters. This ordering is based on a ratio of outgoing dependencies divided by the incoming dependencies. This ratio is calculated for every cluster. The clusters are ordered downwards from the top to the bottom within the graph visualisation component applying this ratio. The ordering mechanism is refreshed when the clustering process is triggered with a new or different java project. Furthermore, the ordering can be executed manually by a right click on the visualisation component and the execution of the 'Rearrange Graph' menu item.

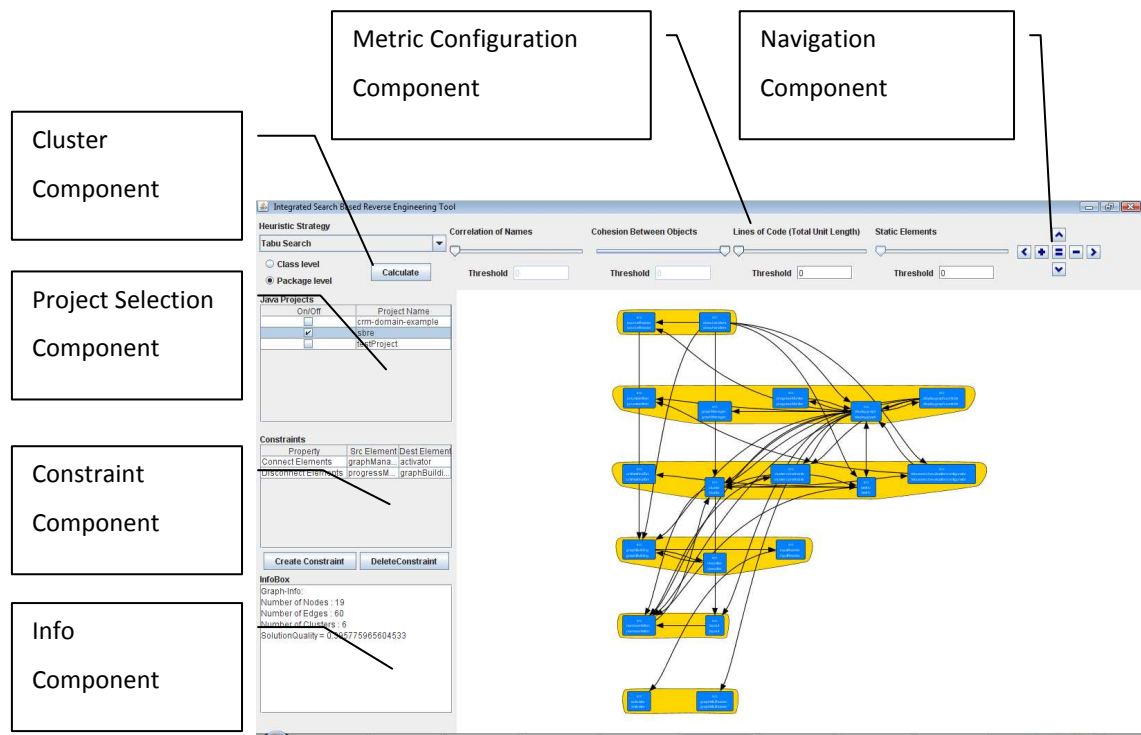


Figure 4.16 : Screenshot of the SBRE component

4.6 Summary

This chapter described the design and implementation of the SBRE component. This component enables the application of greedy algorithm and tabu search in the area of software clustering. Based on the preliminary literature review and guided by the research objective three different frameworks emerged. These frameworks are a *MetricEngine* for the measurement of similarities between artefacts and artefact dependencies, a *ConstraintEngine* for the administration of user constraints and verification of aligning solutions and a *ClusterEngine* for the exploration within the search space and creation of solutions. Additionally, the design and implementation of the solution representation *SBREGraph* is portrayed, which allows the efficient duplication and evaluation of solutions. The core components of the SBRE have also been illustrated from a user point of view.

5 Evaluation of the SBRE component

This chapter describes the evaluation of the SBRE component that has been developed in the course of this research. The evaluation phase reveals the utility of the constructed component and delivers the data to answer the formulated research questions. Regarding this, it is important that a sufficient and appropriate method for the evaluation of the research objective is chosen. As described within the research design chapter, and in alignment with the constructive and explorative character of this research, the execution of experiments is selected as the method to be employed in this study. Section 3.3 already developed a mapping of the conducted experiments to the research questions. The next section describes the design of and environment for the experiments conducted to evaluate the SBRE component.

5.1 Experiment Design

The main purpose of this research is to examine the potential quality contribution of applying a user directed SBSE based software clustering approach. The application of the clustering evaluation methods introduced in Mitchell (2002) and Anquetil and Lethbridge (1999b) to evaluate the quality of clustering are not applicable here. This is due to the greater flexibility of the solution generation approach adopted within this research project. The cluster evaluation approaches from Mitchell (2002) and Anquetil and Lethbridge (1999b) assume a perfect solution as a mix of all applied cluster algorithms and evaluate the distance to this perfect solution for the evaluation of a cluster approach. This implies that all cluster approaches aim for this perfect solution and variations are founded in the deficiency of the cluster algorithms itself. In contrast, the approach within the SBRE component follows the philosophy that a multiplicity of optimal solutions exists. Hence it rather depends on the user needs and the desired perspective of the software system as to which solution is most appropriate. As a result, the formal evaluation mechanisms of Mitchell (2002) and Anquetil and Lethbridge (1999b) are not applicable for the evaluation of the SBRE research component. Instead, an evaluation of the clustering results by a system expert is applied within this research.

To evaluate the cluster results in this way it is necessary to select one or more known software systems for indicative analysis. As the evaluation of the delivered results have an interpreted character it is important that the evaluator in this case has a good knowledge of the design of the analysed software system. Fairly naturally, this motivates the selection of the source code of the SBRE component itself for the evaluation analysis, as the design and components of the SBRE system are well-described during the previous chapters of this research project. The SBRE software system represents a small to middle sized software project, comprising 163 classes, 352 dependencies between these classes, 18 packages and 48 dependencies between these packages. The analysis of the SBRE system therefore enables conclusions to be drawn for software systems of a similar size. Additionally, the publicly available “*crm domain example*” project is used in some experiments as a second, slightly bigger, software system to broaden the results and provide a basis for comparisons.

Aligned with the research objective, the evaluation consists of four different sections. These sections represent the evaluation of the fitness function, the evaluation of the performance of the implemented cluster algorithms, the evaluation of the clustering capability of the SBRE tool and the comparison of the *Barrio* and *Bunch* component with the SBRE component.

5.2 Evaluation of the Fitness Function

An important component of this work is the fitness function as it evaluates the solution candidates and identifies the most feasible solution. The fitness function within this research is called *SolutionQuality (SQ)*. If the *SolutionQuality* measurement is not actually able to distinguish the quality of a solution, the approach has limited merit. Accordingly, the evaluation of the fitness function does not contribute directly to the answer of one of the research objectives. It rather provides a necessary basis for the further evaluation of the research objectives.

The *SolutionQuality* calculates a numerical value, which enables the solutions to be ordered and compared. A comparison of two such measurements is only sensible, however, if they feature the same metric weighting and analyzed the same software system, as the weighting of the metrics and the composition and dependencies of the

software system influence the calculation of the *SolutionQuality* measurement. As a result, it is not possible to derive an absolute quality-estimation depending on the measurement itself. The exchange of the cluster algorithm is unproblematic as the *SolutionQuality* evaluates a solution irrespective of the cluster algorithms. Thus the aim within this experiment is to determine if the *SolutionQuality* measurement enables the identification of good solutions regarding the SBRE tool configuration. In other words, the *SolutionQuality* measurements align with the quality difference of the actual solutions. As the metric configuration defines the behaviour of the fitness function, the metric configuration has to be considered in order to consistently examine the quality of the identified solutions.

An experiment is therefore conducted in which existing solutions are deliberately deteriorated by the inclusion of user constraints, which determine a qualitatively bad cluster landscape. A risk of this experiment is that the inclusion of constraints influences the complete solution. A constraint, which has a negative impact on a part of the solution can still guide the search into another area of the search space, where an improved solution may be found. To minimize this risk the clustered artefacts of the SBRE system are reduced to the four packages *cluster*, *metric*, *cluster.constraint* and *jnumberfield*. Artefacts can be excluded by pressing the right mouse button onto an artefact and choosing the menu item “Ignore artefact ... in next analysis”. The configured *Exclude Patterns* are displayed in the *Constraint Component*. This experiment is conducted with full weight on the *CohesionBetweenObjects* metric and the analysis is conducted on package level. The other metrics are fully disabled and not considered during the cluster analysis. This configuration rewards solutions that combine artefacts in clusters with a high cohesion. Figure 5.1 illustrates the initial result of the clustering without deteriorating constraints and the configuration of the SBRE component.

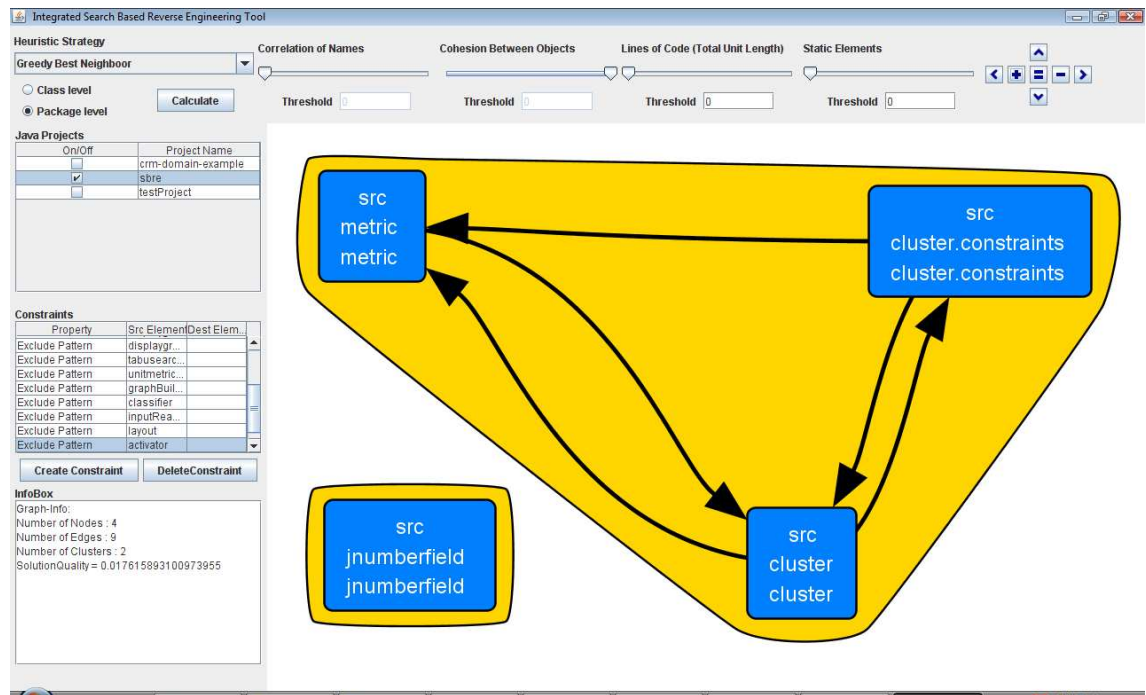


Figure 5.1 : Initial solution of the fitness function experiment

The *metric*, *cluster* and *cluster.constraints* packages feature many dependencies between each other. Regarding this and the concentration on the *Cohesion Between Objects* metric this solution can be considered as feasible.

Based on this initial solution, constraints are introduced, which deteriorate the solution. The first constraint forces the cluster algorithm to separate the *cluster* and *metric* packages. Figure 5.2 demonstrates the result of this clustering.

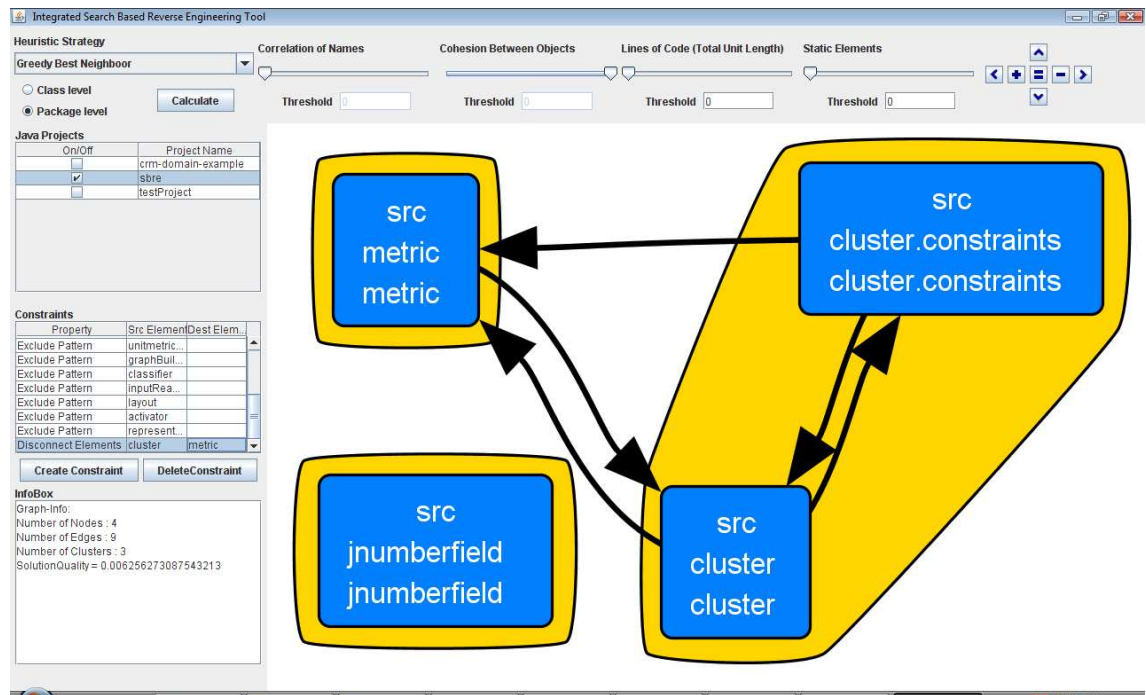


Figure 5.2 : Solution of the fitness function experiment with a separated metric and cluster package

Given the emphasis on the *CohesionBetweenObjects* measurement and the high dependency between the *metric* package and the *cluster* and *cluster.constraints* packages, this solution is less desirable. As illustrated in Table 5.1, this is also expressed in a lower *SolutionQuality* measurement. Within the next clustering the *jnumberfield* package is forced to be included in the same cluster as the *cluster* package. The result of this clustering is illustrated in Figure 5.3.

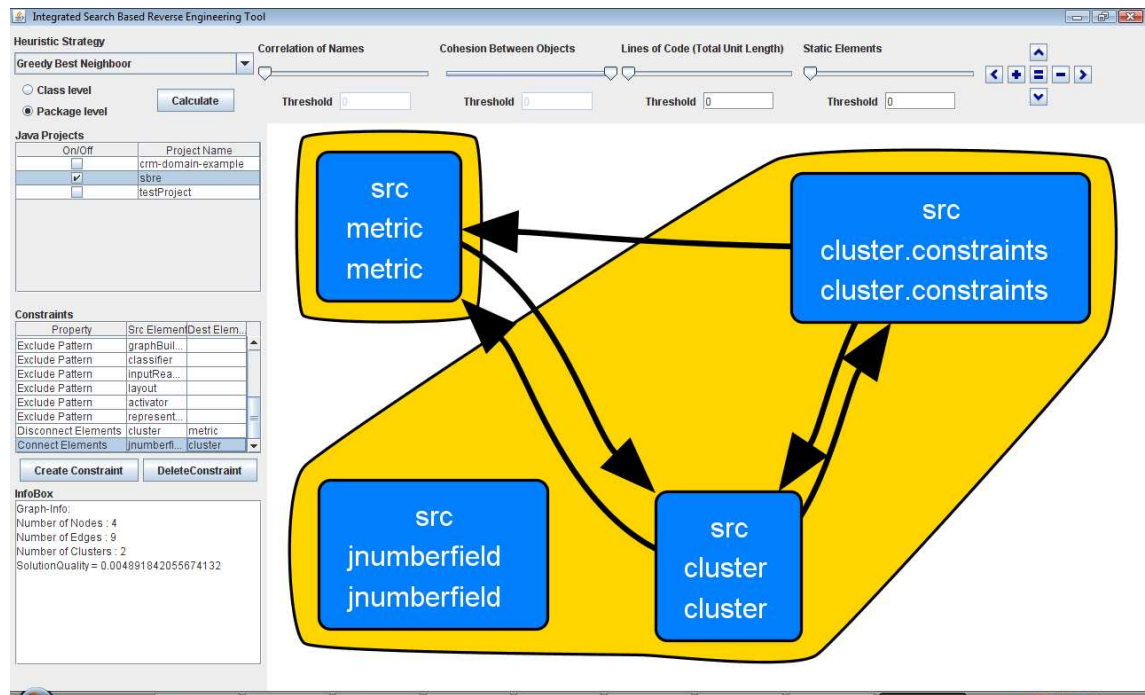


Figure 5.3 : Solution of the fitness function experiment with combined *cluster* and *jnumberfield* package

Again taking into account the focus on the *CohesionBetweenObjects* metric and the missing dependencies of the *jnumberfield* package to the *cluster* and *cluster.constraints* package, this solution can be seen as the most infeasible of those considered. These observations align with the measurements of the *SolutionQuality* fitness function. Table 5.1 illustrates the results of the fitness function experiment. It can be observed that the solution quality is reduced with the inclusion of deteriorating constraints.

Active Constraints	SolutionQuality
no deteriorating constraints	0.0176
cluster and metric package separated	0.0062
cluster and metric package separated \wedge jnumberfield and cluster package combined	0.0048

Table 5.1 : Solution quality values of the fitness function experiment

While the significance of this simple experiment is certainly limited, it serves to demonstrate that a forced deterioration also causes a reduction of the measured *SolutionQuality*. This indicates that a connection exists between the quality of the solution, from a software design point of view, and the *SolutionQuality* measurement.

5.3 Evaluation of the Cluster Algorithms

Within this research the two SBSE based algorithms *GreedyBestNeighbour* and *TabuSearchStrategy* have been designed and implemented. This section reports on the evaluation of the performance and quality of these algorithms. This section contributes especially to the evaluation whether the tabu search concepts are applicable in the area of software clustering. The *GreedyBestNeighbour* algorithm is used as a benchmark. Therefore an evaluation of the *GreedyBestNeighbour* is initially conducted.

The *SolutionQuality* fitness function measurement, which has been introduced in section 4.4.2, remains as a measurement to estimate the quality of a solution and also enables the user to compare the quality of the different cluster algorithm strategies. Certainly, this approach could be seen as subjective as the *SolutionQuality* is also applied as the fitness function within this research to evaluate solutions. However, as the *SolutionQuality* fitness function has been evaluated in section 5.2 and has been shown to be able to distinguish the quality of solutions, the application of the *SolutionQuality* fitness function is, in the opinion of the researcher, meaningful. Other SBSE-based research also relies on the application of the fitness function to evaluate and compare approaches (Jiang et al., 2007; Mitchell, 2002; Seng et al., 2005). All performance experiments described below, which explore the runtime of the algorithms, were conducted twenty five times and the average of the relevant measurements calculated. This should minimize variations caused by the execution of other programs (e.g. virus scans). Further efforts have been taken to minimize these influencing factors with the researcher not working at the test machine during the test runs and disabling unnecessary programs.

5.3.1 Evaluation of the GreedyBestNeighbour Algorithm

This section describes the evaluation of the *GreedyBestNeighbour* algorithm. The only input variable for the greedy algorithm is the analyzed software system. As described previously, the comparison of the solution quality of different software systems is not valid as the design and size of the software system has a direct influence on the calculation of the *SolutionQuality*. Given this, the quantitative evaluation is restricted

to the effect on the runtime of the algorithm regarding the analysis of differently sized software systems.

Runtime Evaluation of the GreedyBestNeighbour Algorithm

Based on the body of the algorithm the computational complexity of the *GreedyBestNeighbour* can be determined as $O(n^2)$, where n is the number of artefacts within the software system. The runtime of the clustering process is n as any node is accessed once and the best move is chosen from the remaining node set. Furthermore, every node is used as a start point for the clustering iteration. The algorithm is additionally influenced by the number of dependencies within the system, as every in- or outgoing dependency of a node is checked to select the best move. To illustrate the computational complexity of the *GreedyBestNeighbour* algorithm, two software systems of different sizes are clustered: the SBRE system and the publicly available “*crm domain example*”. The analysis for both systems is undertaken at both class and package level. In both cases a SBREGraph instance is handed to the algorithm instance, where the SBRENode instances contain either packages or class information. Regarding this, the analysis at different levels does not affect the runtime characteristic. In Table 5.2 the results of this analysis are illustrated. The measured runtime in milliseconds contains only the cluster process of the *GreedyBestNeighbour* algorithm from the moment where the *cluster()* method is executed until the return of the best identified solution. The measuring of the metric values, conversion into different graph models and visualization of the graph is excluded from these measurements.

Analyzed System	Number of Nodes	Runtime in Milliseconds (\emptyset of 25 runs)
SBRE (Package Level)	20	38
CRM (Package Level)	46	708
SBRE (Class Level)	163	12710
CRM (Class Level)	192	40645

Table 5.2 : Runtime in milliseconds of the *GreedyBestNeighbour* algorithm

Figure 5.4 illustrates the progression of the *GreedyBestNeighbour* algorithm runtime in milliseconds relating to the number of input artefacts.

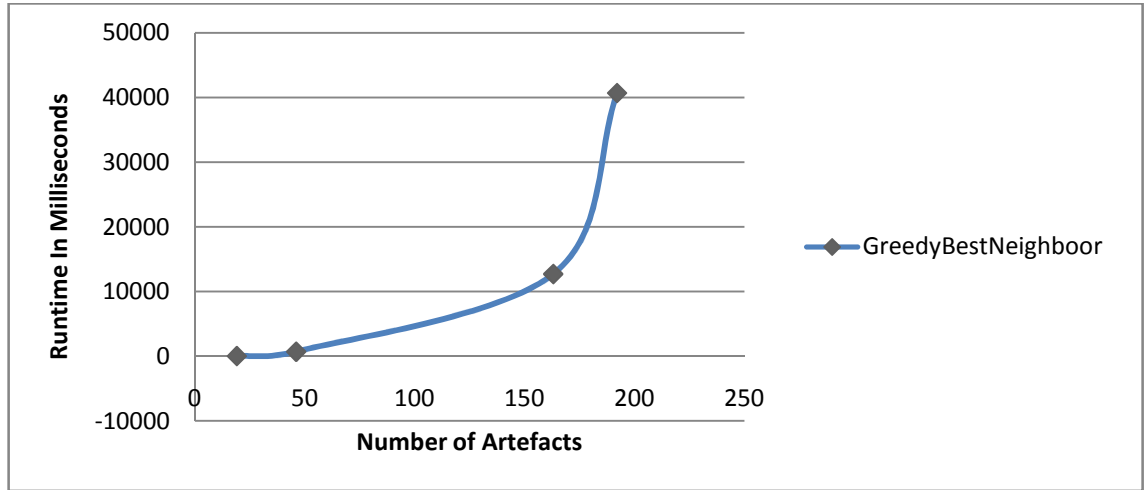


Figure 5.4 : Runtime of the *GreedyBestNeighbour* algorithm depending on the artefact input size

As assumed in the previous section and reflected in Figure 5.4 the runtime of the *GreedyBestNeighbour* algorithm develops within the upper boundary of $O(n^2)$. An interpretation of the delivered solutions is not possible at this stage as the basis for comparison in this analysis is missing. The analysis of the solution quality is conducted in combination with the *TabuSearchStrategy* analysis in the next section.

5.3.2 Evaluation of the *TabuSearchStrategy* Algorithm

In contrast to the *GreedyBestNeighbour* algorithm the *TabuSearchStrategy* algorithm features a wider range of input parameters. These input parameters are the length of the tabu list, the number of iterations until the algorithm terminates, the frequency of the diversification to guide the search into unexplored areas of the search space, the frequency of the intensification to guide the search into the most promising areas and finally the size of the analyzed software system. Each of these parameters has an influence on the runtime of the algorithm, the search and solution improvement process and finally the solution quality. To evaluate the applicability of the *TabuSearchStrategy* algorithm, which is addressed by the second research question, these parameters and their influence on the performance and solution quality of the *TabuSearchStrategy* algorithm have to be examined separately. During this evaluation process a component is developed which enables the tuning of the parameters of the *TabuSearchStrategy*. This component is introduced in the next section.

Implementation Tabu Search Evaluation

The *Heuristic Strategy* selection field of the SBRE component supports the *TabuSearchEvaluation* strategy. Additionally to the algorithmic functionality of the *TabuSearchAlgorithm*, the *TabuSearchEvaluation* strategy features a data input view, which allows the tuning of the input parameters of the tabu search and enables the user to run it multiple times with ascending parameter attributes. The configuration window is illustrated in Figure 5.5.

Maximum Tested Solutions			Length Tabu List		
From	Until	Step Size	From	Until	Step Size
1000	1000	1	40	40	1

Idle Intensification			Idle Diversification		
From	Until	Step Size	From	Until	Step Size
100	100	1	100	100	1

Ok Cancel

Figure 5.5 : Configuration of the *TabuSearchStrategy* evaluation component

The four parameters *Maximum Tested Solutions*, *Length of the Tabu List*, *Idle Intensification* and *Idle Diversification* can be tuned. Every parameter is split into three values, the “from” input value defines the start value of the search, the “until” value determines the termination condition and the last tested configuration, and finally the step size defines the changing interval. As a consequence the search is executed n times, with $n = (\text{“until input value”} - \text{“from input value”}) / \text{“step size”}$. The iterations of the cluster analysis are executed independently and build up respectively on the best identified solution of the *GreedyBestNeighbour* algorithm analysis. The result of each iteration is displayed in the *Eclipse Console*.

Table 5.3 illustrates the result of a single iteration with one parameter configuration. Besides the *TabuSearchStrategy* algorithm parameter configuration for the current run, the table also shows the runtime in ms, the *SolutionQuality* of the best found solution, the number of rejected solution candidates and the number of diversification and intensification runs.

Number of Intensifications	7
Number of Diversifications	10
Tabulist Rejects	8
Solution Quality	0.54
Runtime in ms	397
Idle Diversification	100
Idle Intensification	150
Length Tabu List	40
Maximum Tested Solutions	467

Table 5.3 : Example of one result of the *TabuSearchStrategy* algorithm evaluation

The *GreedyBestNeighbour* algorithm delivers within the conducted experiments the initial graph for the *TabuSearchStrategy* algorithm analysis, which is also the default behaviour of the *TabuSearchStrategy* algorithm. However, the time measurements of the *TabuSearchEvaluation* component do not include the runtime of this *GreedyBestNeighbour* algorithm run. The reason for this is, that the *TabuSearchStrategy* algorithm allows the employment of previously created solutions as the initial solution (see also: Figure 4.15). This motivates that any solution could be utilized as the initial graph for the *TabuSearchStrategy* algorithm, this solution can be created by any algorithm or even created or manipulated by the interaction of user. As the time effort to create the initial solution cannot be determined, the runtime measurements of the *TabuSearchStrategy* algorithm are specified without the creation of an initial solution.

As described previously the *TabuSearchStrategy* algorithm is influenced by the “Idle Diversification”, the “Idle Intensification”, the “Maximum Tested Solutions” and the “Length of the Tabu List” parameters. It should also be considered that the individual parameters have an influence on each other. Regarding this, the conducted experiments have the aim to illustrate that the individual parameters influence the behaviour of the algorithm. Additionally, it should be shown if the tabu search algorithm is able to improve the search in comparison to the output of the *GreedyBestNeighbour* algorithm. The aim of these experiments is not to derive general assumptions about the configuration of the tabu search, rather it should provide some

degree of evidence as to whether the tabu search algorithm is applicable in the field of software decomposition. Where it is possible and appropriate to generalize a result, a hypothesis is formulated to give an idea for such a generalization. However, the experiments reported here do not have sufficient scope to support or refute these general assertions.

As the metric configuration is not part of this experiment and does not influence the outcome, the metric configuration is untouched for the evaluation of the cluster algorithms. This means that every metric is defined with a weight of 0.5, where 1.0 would be the maximum weight. Figure 5.6 illustrates the metric configuration used during the following experiments.

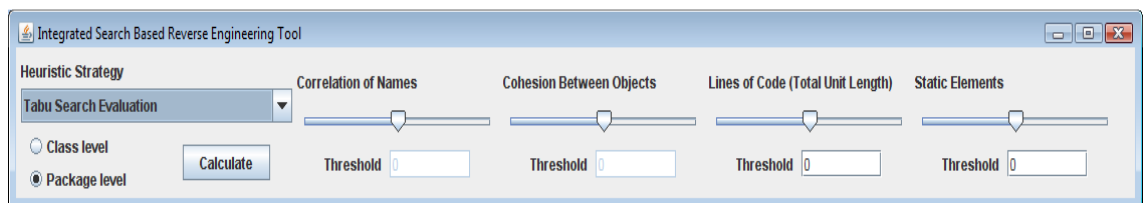


Figure 5.6 : Illustration of the metric configuration during the cluster algorithm evaluation

To examine, if the *TabuSearchAlgorithm* is able to improve the identified solution of the *GreedyBestNeighbour* solution, the following experiment is conducted. It examines the change of the *SolutionQuality* regarding the increase of the tested solution candidates.

Evaluation Maximal Tested Solutions

The termination of the tabu search clustering algorithm is determined by the number of maximal tested solution candidates. The number of maximal tested solutions influences the *SolutionQuality* and the runtime of the algorithm. Within this experiment the number of maximal tested solutions is increased from 1 until 1000 in single unit increments. The accompanying parameters are the length of the tabu list and the number of iterations without improvement until the activation of the diversification algorithm. The tabu list is confined to a maximum length of 40 entries and the diversification and intensification processes are triggered after 100 iterations without an improvement of the solution quality. The following experiments within this section will show that this configuration is optimal for the intensification and

diversification parameterisation to obtain the best *SolutionQuality* measurement. The excessive length of the tabu list prohibits the confinement in an infinite loop. Figure 5.7 displays the parameter configuration of the conducted experiment.

Maximum Tested Solutions			Length Tabu List		
From	Until	Step Size	From	Until	Step Size
1	1000	1	40	40	1

Idle Intensification			Idle Diversification		
From	Until	Step Size	From	Until	Step Size
100	100	1	100	100	1

Ok Cancel

Figure 5.7 : Configuration of the maximal tested solutions experiment

This experiment aims to evaluate the performance of the *TabuSearchStrategy* algorithm to improve the *SolutionQuality* measurement and the impact on the runtime in comparison with the *GreedyBestNeighbour* algorithm. Therefore the experiment contributes to the evaluation of the applicability of the *TabuSearchStrategy* in the area of software clustering.

The experiment is conducted with the SBRE system at both package and class level, which enables a degree of analysis of different system sizes. Figure 5.8 and Figure 5.9 illustrate the results as the development of the *SolutionQuality* measurement relating to the number of maximal tested solutions.

Figure 5.8 pictures the analysis of the SBRE system at the package level and represents a small system with 20 SBRENode instances and 56 SBREEdge instances.

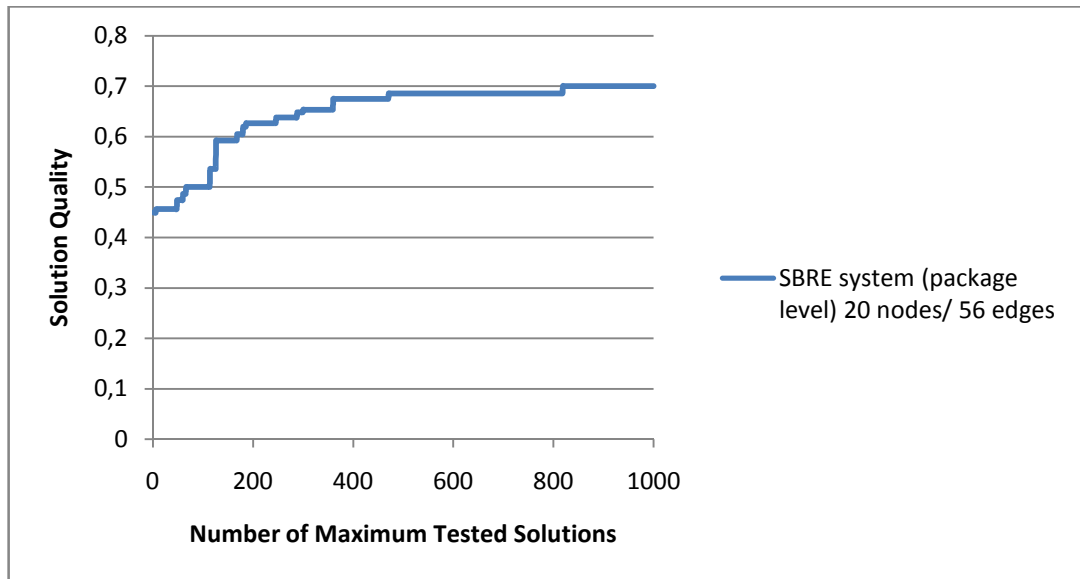


Figure 5.8 : Solution quality in relation to maximum algorithm iterations of the SBRE system (package level)

Figure 5.8 illustrates a continuous improvement regarding the *SolutionQuality* based on the best identified solution of the previous *GreedyBestNeighbour* algorithm analysis. A maximum level of solution improvement will be obtained after a certain number of iterations. No further improvement of the *SolutionQuality* will be possible with the current parameter configuration. To illustrate the performance of the tabu search algorithm with bigger systems, the SBRE system is also clustered at the class level. In this case the *SBREGraph* comprises a system with 163 nodes and 352 edges. Figure 5.9 demonstrates the development of the solution quality in regard to the increased number of maximum tabu search iterations for the SBRE system analysis at the class level.

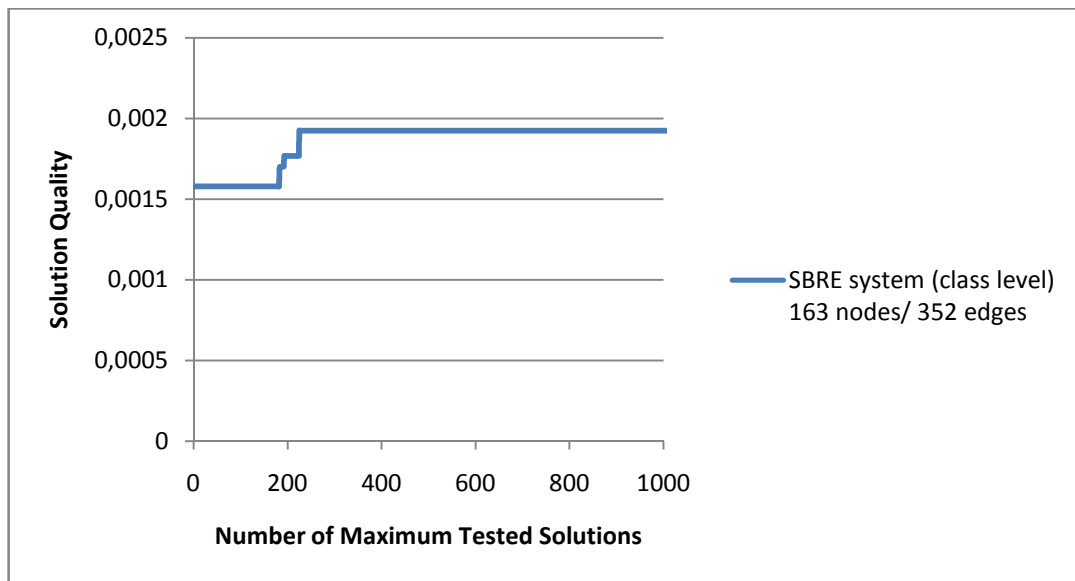


Figure 5.9 : Solution quality in relation to maximum algorithm iterations of the SBRE system (class level)

The plateau and lack of improvement from iteration 250 onwards can be caused by a variety of reasons, for example, a too short tabu list and, as a result, confinement in an infinite loop. A deficiency in the diversification or intensification triggering could also force or confine the search in a non-improving area of the search space. Also an insufficient selection of solution candidates of the tabu search algorithm itself could lead the search into non-improving areas of the search space. Finally, the reason for the plateau could also be that an optimal solution within the search space has been identified and no further improvement is possible with the current configuration. As it is not central to this work the reasons for this plateau have not been further investigated.

Figure 5.10 assesses the runtime of the *TabuSearchStrategy* and *GreedyBestNeighbour* algorithm regarding “the number of artefacts” within the software system. As data basis is the SBRE system and the „*crm domain example*“ project used. Both projects are clustered on class and package level to broaden the data base. The clustering with the *TabuSearchStrategy* run is conducted with 100 and 1000 as the “number of maximal tested solutions”. The measurement of the *TabuSearchStrategy* runs do not include the runtime of the *GreedyBestNeighbour*, which is executed by default if no other initial solution is supplied. If the default behaviour is applied, the runtime of the *GreedyBestNeighbour* function would have to be added on top of the both

TabuSearchStrategy runtime functions. Considering this, the runtime of the *TabuSearchAlgorithm* is higher than the runtime of the *GreedyBestNeighbour* algorithm.

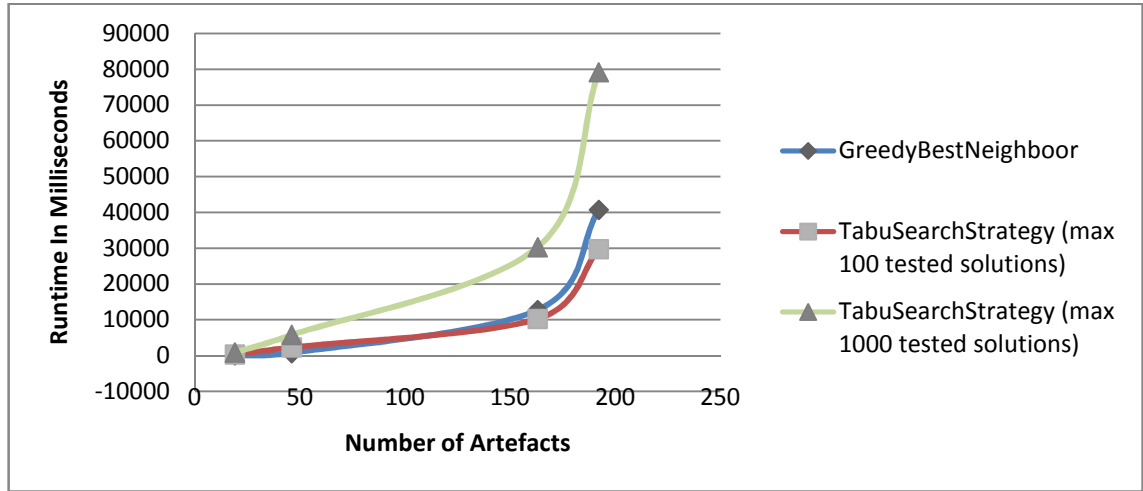


Figure 5.10 : Algorithm runtime in relation to number of artefacts

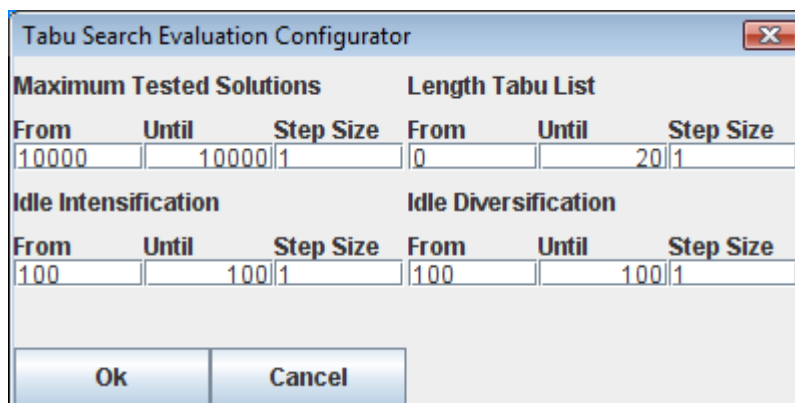
Figure 5.10 illustrate that the *TabuSearchStrategy* algorithm exhibits a runtime, which is smaller than a polynomial function. This contributes to the applicability of the *TabuSearchStrategy* in the field of graph partitioning, which is a NP hard problem.

In conclusion, this section illustrated the impact of the variation of the maximal tested solutions on the *SolutionQuality*. It has been observed that the *TabuSearchStrategy* algorithm is able to improve the delivered solutions of the *GreedyBestNeighbour* algorithm. Furthermore, it could be shown that the *SolutionQuality* increases with the number of tested solution candidates, but also the runtime of the solution finding process increases. It depends, on the requirements of the stakeholders, if the increase of the *SolutionQuality* justifies the longer runtime of the *TabuSearchStrategy*.

The *TabuSearchStrategy* algorithm features additional input variables, which have an influence on the search process. The previous section assumed a fixed length of the tabu list and of the idle diversification and intensification iterations. It is of importance for the evaluation for the first research objective whether the length of the tabu list influences the solution finding process. The next experiment illustrates the consequences of variations of the tabu list length on the *SolutionQuality*.

Evaluation Length Tabu List

The entries of the tabu list help to prevent solution cycles confining the search in an infinite loop. The following experiment examines the influence of the tabu list length on the performance on the *SolutionQuality*. Thence the experiment contributes to the evaluation of the applicability of the *TabuSearchStrategy* algorithm in the area of software clustering. Figure 5.11 illustrates the configuration of the parameters. It can be assumed that a high number of tested solutions also increase the chance of duplicates. To combat this, the maximum tested solutions are increased to 1000. The idle intensification and diversification values are set at 100. The experiment is realized with these values and an ascending length of the tabu list from 0 to 20. In total 21 experiment runs are conducted. The experiment addresses the effect of the variation for the illustrates that the length of the tabu list has an influence on the *SolutionQuality* measurement.



Maximum Tested Solutions			Length Tabu List		
From	Until	Step Size	From	Until	Step Size
10000	10000	1	0	20	1

Idle Intensification			Idle Diversification		
From	Until	Step Size	From	Until	Step Size
100	100	1	100	100	1

Ok Cancel

Figure 5.11 : *TabuSearchStrategy* configuration of the tabu list length experiment

Figure 5.12 illustrates the development of the solution quality in relation to the increase of the tabu list length. The value at -1 on the x axis represents the initial *SolutionQuality* delivered by the *GreedyBestNeighbour* algorithm.

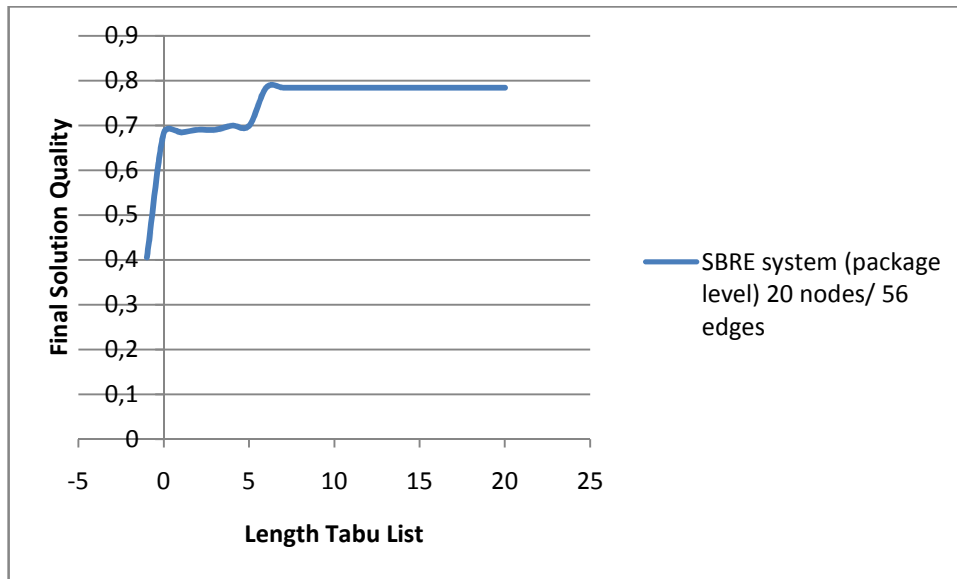


Figure 5.12 : Solution quality in relation to the length of the tabu list

Figure 5.13 illustrates the increasing number of rejected solutions in relation to the length of the tabu list. This increase aligns with the change of the *SolutionQuality* measurement from Figure 5.12.

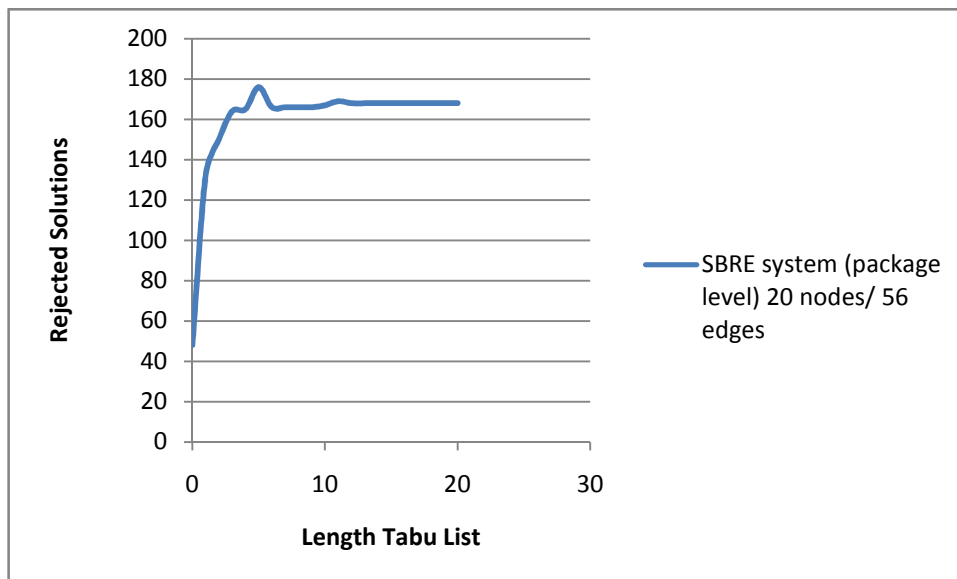


Figure 5.13 : Rejected solution candidates in comparison to the length of the tabu list

The examination of the influence of the tabu list length on the *SolutionQuality* of the SBRE system at the class level discovered no improvement in relation to the increased length of the tabu list. A reason for this could be that the *TabuSearchStrategy* identifies an optimal solution, which is in the direct neighbourhood to the initial discovered solution from the *GreedyBestNeighbour* algorithm. In this case the

TabuSearchStrategy would only need a small number of iterations to find an optimal solution. Another explanation is that the enlargement of the search space reduces the risk to create the same solution again. An argument for this is that the number of rejected solutions did not increase to any noteworthy degree during the experiment process.

In conclusion, it has been demonstrated that the application of the tabu list is able to prevent cycles within the SBRE system. Another finding is that a relatively short tabu list even in combination with a high number of maximal tested solutions can have a significant influence on the solution quality. In relation to the SBRE cluster example a maximal tabu length of six entries is sufficient to prevent the occurrence of cycles within a search of 1000 solution candidates.

Evaluation Idle Diversification and Intensification Iterations

The remaining parameters that control the *TabuSearchStrategy* are the “Idle Diversification” and “Idle intensification” parameters. These parameters feature a similar characteristic. Because of this, the consideration of both is consolidated within this section. The evaluation of these parameters adds to the second research objective whether tabu search is applicable in the field of software clustering. Note that it is not part of the research objective to examine the best parameter configuration to trigger the diversification and intensification process.

The consequence of diversification and intensification is that the search does not proceed in the current area of the search space. Instead, the search is continued in a different area of the search space, which is determined by the intensification or diversification strategy. However, there is of course a chance that the search in the current search area would have identified an improved solution candidate within the next iterations. It cannot be determined in any absolute sense that diversification or intensification will improve or reduce the effectiveness of the search in any general sense. The optimal configuration, for these parameters, depends on the analyzed software system, on the other parameter configurations and also on the requirements of the stakeholders. The triggering of the diversification and intensification at a certain stage of the search can be beneficial where it leads from another state of the search

into improving areas of the search space. However, the objective of this research is to evaluate the application of tabu search in the area of software clustering, and for this it is sufficient to evaluate if the diversification and intensification have an influence on the search. Given this aim, it has to be examined, if the diversification and intensification processes are able to be activated and lead the search into different areas of the search space. This would be sufficient to demonstrate the successful application of diversification and intensification strategies within the SBRE tool. The remaining paragraph portrays the results of the experiment to examine the application of the diversification and intensification strategy within the SBRE tool.

The “Idle Diversification” parameter defines the number of non-improving solution candidates to be tested in a row, until the diversification method is started, so as to guide the search into an unexplored area of the search space. From a hypothetical point of view there is an expectation that too low an idle number of iterations would not allow the algorithm to explore the area of the current search space adequately in order to identify improving solutions, but too high an idle number is also not beneficial as the algorithm could spend too long in non-improving areas of the search space. Two experiments with the SBRE system (at the package level) have been conducted to examine the influence of the diversification start point on the algorithm runtime and solution quality. Figure 5.14 illustrates the configuration of the first experiment.

Maximum Tested Solutions			Length Tabu List		
From	Until	Step Size	From	Until	Step Size
100	100	1	100	100	1

Idle Intensification			Idle Diversification		
From	Until	Step Size	From	Until	Step Size
1000	1000	1	2	150	1

Ok Cancel

Figure 5.14 : *TabuSearchStrategy* configuration for the diversification experiment

The first experiment is conducted with a maximum of 100 tested solutions. The second experiment tests 1000 solution candidates per experiment run. Referring to the previous tabu search experiment, it is important that the length of the tabu list is

sufficient to not bias the data collection caused by confinement in an infinite loop. The “Idle Diversification” parameter is increased from 2 until 150 during each experiment. Respectively, 149 independent experiment runs are conducted for each of the both experiments. As a consequence is the number of diversification runs reduced by the increasing of the “Idle Diversification” parameter.

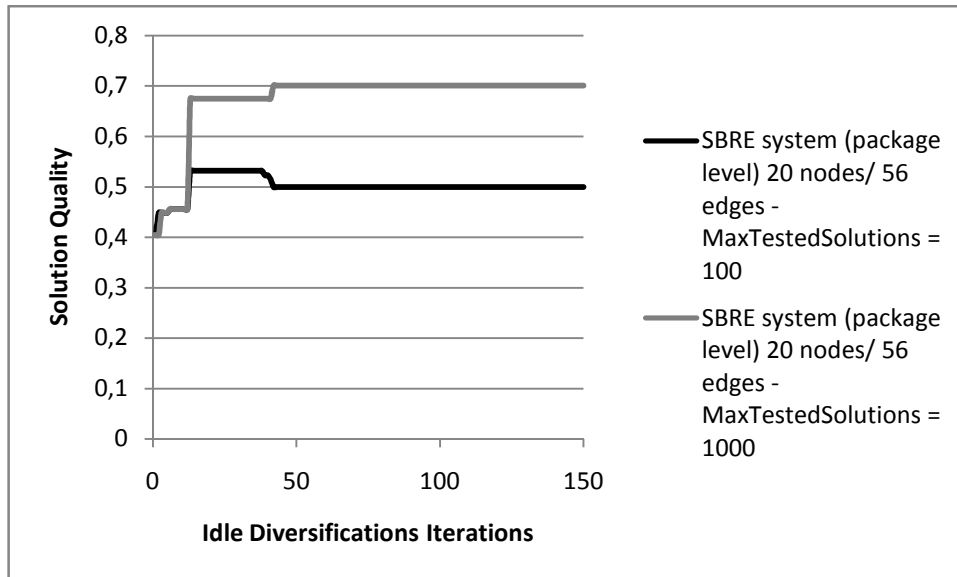


Figure 5.15 : Solution quality in relation to idle diversification iterations (package level)

In line with the experiment results are the number of triggered diversification executions illustrated in Figure 5.16.

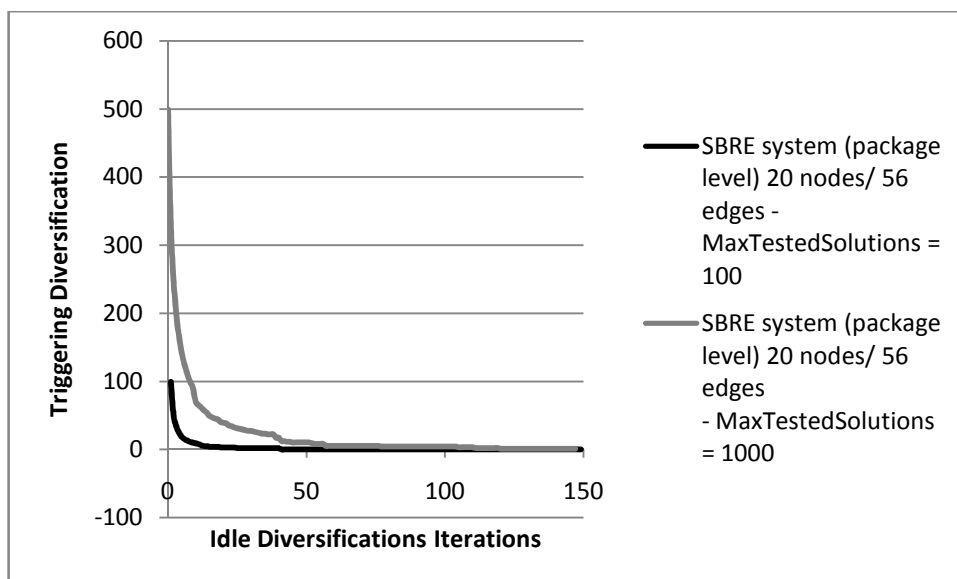


Figure 5.16 : Number of triggered diversification runs in relation to idle diversification iterations

Based on these experiment results, it can be stated that the triggering of the diversification process changes the *SolutionQuality*. Furthermore, a more frequent triggering of the diversification reduces the *SolutionQuality*, which aligns with the expected results. However a rare execution of the diversification leads, within this example, to only a slight deterioration in the 100 maximal tested solution example. The experiment with a maximum of 1000 tested solutions finds even the optimal results when the diversification is not started during the search process. These results can be a coincidence in combination with the analysed software system or an insufficient design of the diversification method. Based on this, the same experiment is conducted at the class level of the SBRE system. The number of maximal tested solution candidates is confined to 1000. Figure 5.17 illustrates the results of this experiment.

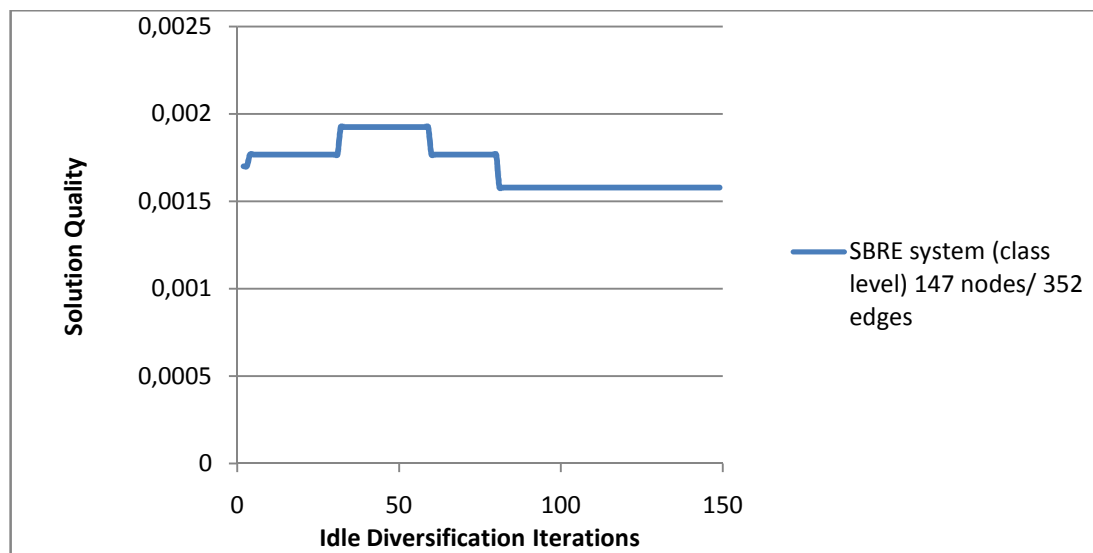


Figure 5.17 : Solution quality in relation to idle diversification iterations (class level)

The results of the analysis of the larger system illustrate that the diversification can indeed guide the search process into more feasible areas of the search space. A hypothesis regarding the different outcomes of the experiments on package and class level could be that the search within the previous experiment, with a maximal number of 1000 tested solution candidates, already achieves a reasonable coverage of the search space without the application of the diversification process. This could explain why the application of the diversification has only a negative effect on the

SolutionQuality. However as stated this is just a hypothesis and the examination of the reasons for this behaviour exceeds the scope of the present research.

Finally as the last parameter the “Idle Intensification” has to be considered. The approach to intensify the search within the search space, by generating a solution that comprises the highest occurrence of artefact combinations, has not improved the identified solutions. The best results are retrieved, when the intensification mechanism is disabled. The collected data of these measurements can be found on the enclosed CD. However, even the deterioration of the frequent application of the intensification shows that the intensification has an effect on the *TabuSearchStrategy* solution finding process. It is noted that the intensification solution is generated and used as the initial solution for the continuing search. The analysis of a different software system could discover that the same intensification measure is applicable and effectuates a *SolutionQuality* improvement

5.4 Evaluation of the SBRE Cluster Analysis

As stated before, the SBRE component does not aim for a perfect solution; rather, it creates different perspectives and views into the software system in keeping with the user configuration and preferences.

The particular strength of this research is that the search algorithm and fitness function adapt to the metric configuration of the SBRE component. To illustrate, the following experiments demonstrate the cluster capability of the SBRE component with a particular focus on the metric configuration of the SBRE component. In addition, the cluster capability of the *TabuSearchStrategy* algorithm is examined. The execution of these experiments demands the application of the main components of the SBRE cluster component. And these components and their interplay are then evaluated. The first component is the metric framework and the weighting of the individual metrics to focus the search on a certain perspective of the software implementation view. Additionally, the application of user constraints to include design knowledge into the cluster process is intended to guide the search into a special area of the search space. Finally, the use of *SolutionQuality* as the fitness function is involved to identify the optimal solution of the current search. The conduct and results of these experiments

are described within the next three sections. The next paragraph examines the capability of the SBRE component to change the cluster outcome in line with different metric configurations.

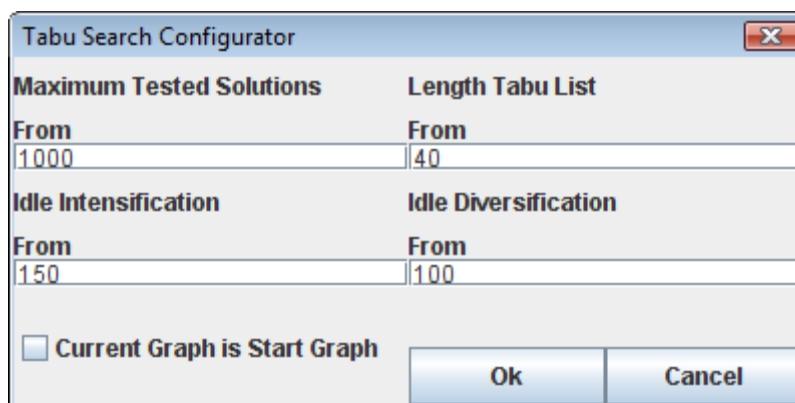
5.4.1 Evaluation of Multiple Implementation Perspectives

This section demonstrates how the application of the metric configuration enables the manipulation of the solution search process. The conducted experiments contribute to the evaluation whether a multi metric approach is able to create different implementation perspectives and therefore contributes to the third research question. These experiments are divided into two parts. The first experiment aims to examine the effect of the configuration of the CoN and CBO metric to derive different cluster landscapes of the structure of the software system. The second section evaluates the capability of the multi metric approach to identify clusters, which feature a certain degree of deterioration.

It is not the intent of it to determine whether one particular solution is better than another. This depends on the design of the metrics, the perceptions of the user and also the design of the analysed software system. Rather, the intent is to show if the approach is able to produce different cluster landscapes depending on the metric configuration of the SBRE component.

Derivation of a structure using the CON and CBO metrics

To conduct this experiment the *TabuSearchAlgorithm* is applied to create the solutions. The configuration of the *TabuSearchAlgorithm* is illustrated in Figure 5.18.



Maximum Tested Solutions	Length Tabu List
From 1000	From 40
Idle Intensification	Idle Diversification
From 150	From 100

☐ Current Graph is Start Graph

Ok Cancel

Figure 5.18 : Configuration of the *TabuSearchStrategy* algorithm within the multiple view experiment

This first experiment run is conducted by applying the full metric weight onto the *CBO* metric whereas the second experiment run is conducted by applying the full metric weight onto the *CON* metric. The expected result is that the cluster landscape should change for the two runs. Additionally, the clusters in the second run should comprise artefacts that feature a high correlation of names. Figure 5.19 illustrates the result of the experiment. The cluster results are also displayed in Table 5.4.

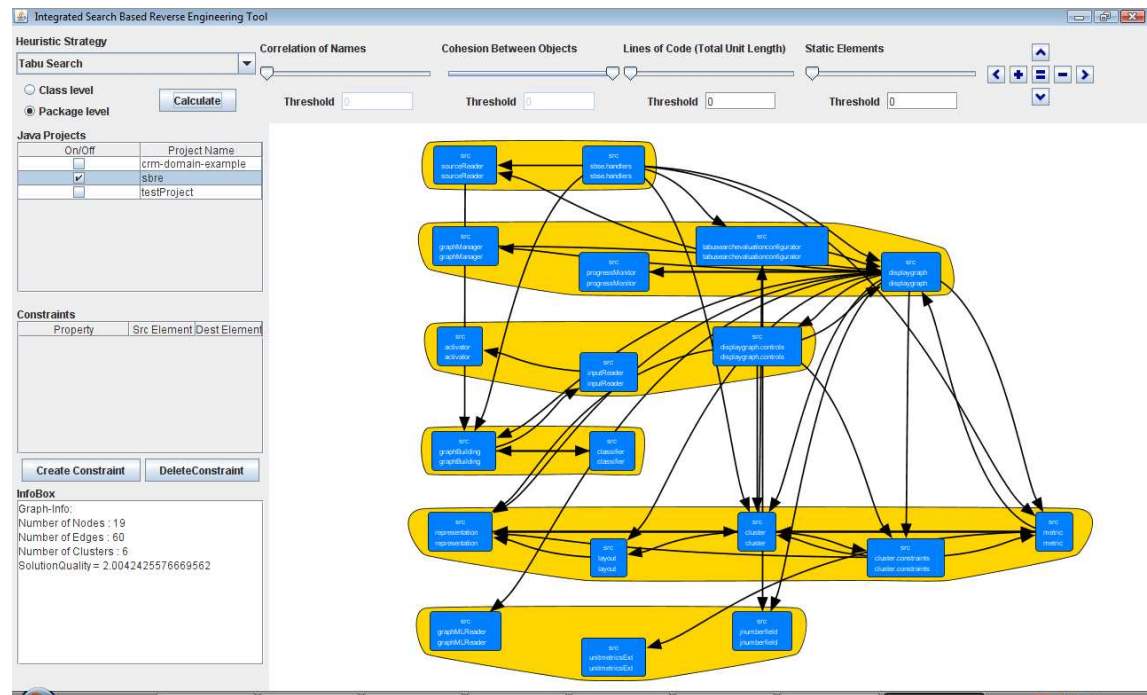


Figure 5.19 : Result of the multiple view experiment with 100% weight on the CBO metric

The second experiment is conducted with the same *TabuSearchAlgorithm* configuration. The full weight of the *CON* metric is enabled and the *CBO* metric is disabled for this experiment run. The result of the second experiment run is illustrated in Figure 5.20. The positions of the artefacts have not been changed during the clustering. This implies that every artefact is visualised at the same position of the graph and therefore the changes in dependencies and clusters are more visible. A comparison of the cluster results of the previous two experiments is also visualized in Table 5.4.

artefacts are combined in a cluster, yet they feature only low name similarity. Another reason for this obfuscation is the penalizing of heterogeneous clusters. The fitness function penalizes small and very large clusters to support a homogeneous cluster landscape. This behaviour is beneficial and necessary to deliver simple cluster landscapes. If a system is not designed following the corresponding metric configuration, the penalizing strategy obfuscates the clustering result and enables solutions that do not represent the configured cluster results.

Full Weight - CBO Metric	Full Weight - CON Metric
Cluster0 cluster cluster.constraints layout metric representation Cluster1 activator graphMLReader inputReader Cluster2 classifier graphBuilding Cluster3 displaygraph.controls jnumberfield unitmetricsExt Cluster4 displaygraph graphManager progressMonitor tabusearch- evaluationconfigurator Cluster5 sbse.handlers sourceReader	Cluster0 displaygraph displaygraph.controls graphManager Cluster1 activator jnumberfield progressMonitor representation Cluster2 classifier graphBuilding inputReader unitmetricsExt Cluster3 cluster cluster.constraints layout metric Cluster4 graphMLReader sbse.handlers sourceReader tabusearch- evaluationconfigurator

Table 5.4 : Comparison clustering solution with 100% weight on the CBO or CON metric

The design of the SBRE system has not followed a strict name policy, hence the reason for the poor cluster result of the *CorrelationOfNames* analysis of the SBRE system. To support this assertion the „*crm domain example*“ project is also clustered. The „*crm domain example*“ follows a clear naming policy. The previous *TabuSearchStrategy* displayed in Figure 5.20 is applied. Because of the size of the cluster landscape only an

extract of the CON clustering result is displayed in Figure 5.21. The complete result can be reproduced applying the SBRE tool or the cluster result can be found as file on the enclosed CD. The outcomes of this analysis showed that the clustering landscape changed, depending on whether the metric configuration focused on the CBO or CON metric. The clustering results of the „*crm domain example*“ with a focus on the CON metric reflect a high name correlation within the artefacts of the individual clusters. This illustrates that the successful application of a certain metric configuration depends also on the design of the analysed software system. If the design of the analysed software system does not support the attributes of the metric configuration, the produced cluster landscape will also be unfeasible.

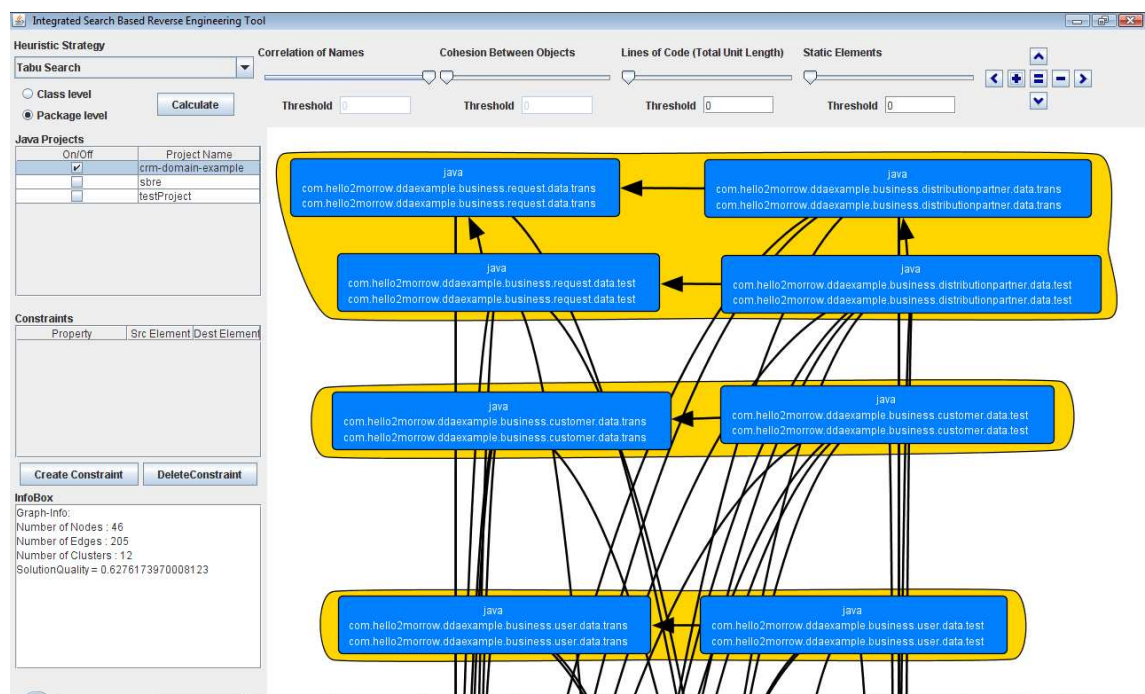


Figure 5.21 : Extract of the CON analysis cluster landscape of the „*crm domain example*“

In conclusion, it has been shown in this experiment that the modification of the metric weighting guides the search into different areas of the search space. It is also evident that other factors beside the metric configuration e.g. design of the metric, design of the fitness function and design of the analysed software system, can all influence the search process and can obfuscate the result.

Identification of Deteriorated Subsystems

This section evaluates if it is feasible to use SBSE-based software clustering in combination with metrics to identify problematic code segments at a high abstraction level. It is important that artefacts that are combined in one cluster feature a dependency connection between one another. Only a dependency between these artefacts acknowledges a design connection between them. This design connection is of relevance as an identified cluster should give an indication of an interconnected subsystem, signalled by a high value for the metric measurement. Only interconnected subsystems are relevant to estimate the strength of a metric violation within a certain part of the software system. The supporting idea is that the refactoring and elimination of two unrelated high metric measurements would be better planned as two independent development tasks. The *GreedyBestNeighbour* algorithm uses the dependencies between artefacts to identify a solution. This favours the *GreedyBestNeighbour* for the implementation of this experiment. Additionally, for the identification of problematic artefacts it is interesting to identify the originating artefacts as concrete as possible. Given this requirement the experiment should be conducted on class level.

The SBRE system provides two metrics that support the identification of deteriorated subsystems. These metrics measure aspects of each artefact: the *Lines of Code* and *Number of Static Elements*.

Not all artefacts are of interest for this view so only artefacts that feature a high metric value should be included. The determination of the metric threshold depends strongly on the expectations of the stakeholders and the intention of the analysis. This experiment focuses on the identification of a high incidence of static elements within an artefact. Figure 5.22 illustrates the configuration and cluster result of the analysis. The metric configuration focuses completely on the measurement of the *Static Elements*. The *Correlation of Names*, *Cohesion Between Objects* and *Lines of Code Metric* are not considered for the creation of the clusters. The threshold of the *Static Element* metric is confined to six. This value has been determined by trial and error

and delivers a clear result for the SBRE system. A lower threshold, for example, can be chosen when these violations are eliminated and higher quality software is aspired.

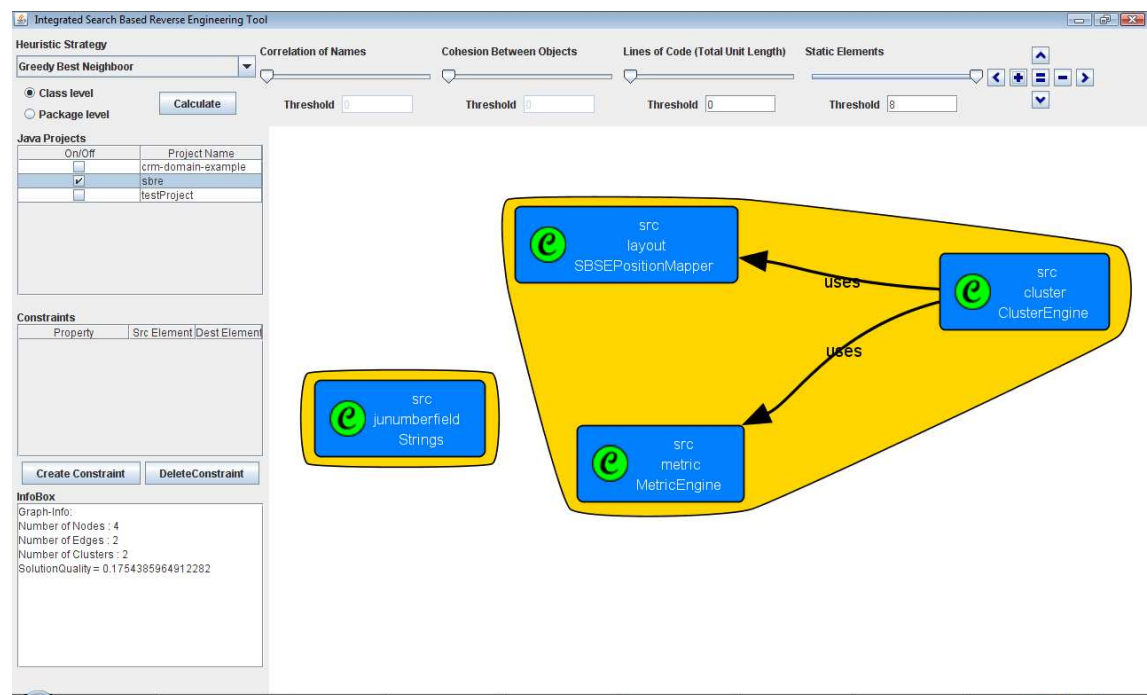


Figure 5.22 : Result of the deterioration cluster analysis with 100% weight on the static elements metric

During this experiment four artefacts were identified as exceeding the defined threshold. The *GreedyBestNeighbour* cluster algorithm combined these artefacts into two clusters. One cluster comprises the *Strings* class of the *jnumberfield* package. This class constitutes a part of an external library, which offers external functionality for the SBRE system. Of greater interest is the bigger cluster which constitutes the *MetricEngine*, *ClusterEngine* and *SBSEPositionMapper*. These classes are of central importance for the SBRE system. As described in sections 4.3 and 4.4.4 the *MetricEngine* and the *ClusterEngine* represent the context classes of the encapsulated frameworks. In this light it is appropriate that these classes are known system-wide. However, instead of offering the structure and behaviour through static elements, one instance of the class should be available system-wide following the singleton design pattern introduced by Gamma, Helm, Johnson and Vlissides (1995). The same argument can be applied to the *SBSEPositionMapper*, which should also be refactored using the singleton design pattern. These actions should abrogate the cluster and the elimination of this cluster should improve the quality of the software system.

It would also be possible to combine different metrics to identify deteriorated subsystems. As described previously the SBRE system also offers the *Lines of Code* metric. The combination of these metrics with thresholds would additionally decrease the number of artefacts to be traversed and concentrate the search for more strongly deteriorated subsystems. Additionally, it is possible to extend the number of available node metrics to change the focus of the search.

In conclusion, it has been shown in this experiment that the application of the *Static Elements* metric can help the user to identify deteriorated subsystems. Based on this knowledge a refactoring can be planned to eliminate these clusters with the intention to improve the quality of the software system.

5.4.2 Evaluation of Rearchitectureing Functionality

The aim of the following experiment is to identify a feasible and suitable structure for a software system after applying the clustering functionality of the SBRE component. This experiment applies the *TabuSearchStrategy* and evaluates therefore the capability of this algorithm to produce meaningful cluster results.

As the analysis focuses on the rearchitectureing of the software system, the *CohesionBetweenObjects* and the *Correlation of Names metric* are relevant. The *CohesionBetweenObjects* is relevant, because the clustering should identify artefacts that feature high cohesion between each other on a design and dependency level. With respect to the correlation of names, it has already been shown that the design of the SBRE system did not follow a strict naming policy. Nevertheless, through trial and error it has been found that a combination of the *CohesionBetweenObjects* and the *Correlation of Names* metrics produces the best result to identify a structure and functional correlating clusters. The identified configuration is illustrated in Figure 5.23. This configuration considers the CON metric with fifty percent and the CBO metric with one hundred percent.

Figure 5.23 shows a configuration interface for metric analysis. It consists of four main sections, each with a slider and a corresponding 'Threshold' input field:

- Correlation of Names:** Slider and Threshold 0.
- Cohesion Between Objects:** Slider and Threshold 0.
- Lines of Code (Total Unit Length):** Slider and Threshold 0.
- Static Elements:** Slider and Threshold 0.

Figure 5.23 : Metric configuration of the rearchitecturing analysis

The metric configuration therefore supports the identification of modules with high cohesion between each other and also takes account of the similarity of names for this modularisation. Utilising this approach, Figure 5.20 illustrates the subsequent configuration of the *TabuSearchStrategy*. The default parameters of the *TabuSearchStrategy* are used, with the exception that the maximal tested solutions are determined with 1000.

Figure 5.24 shows the 'Tabu Search Configurator' dialog box. It contains the following configuration parameters:

Maximum Tested Solutions	Length Tabu List
From 1000	From 40
Idle Intensification	Idle Diversification
From 150	From 100

At the bottom, there is a checkbox labeled 'Current Graph is Start Graph' (which is unchecked) and two buttons: 'Ok' and 'Cancel'.

Figure 5.24 : *TabuSearchStrategy* configuration of the rearchitecturing experiment

Based on this configuration, the visualisation of the cluster analysis is illustrated in Figure 5.25. In addition to the displayed result for the SBRE cluster landscape, textboxes and separators have been superimposed to simplify the discussion and to provide an indication of the comprised functionality within the clusters. These cluster descriptions represent a functional interpretation of the system, but should not be misinterpreted as a layering of the system or architecture definition of the system. This interpretation is conducted by the developer of the SBRE system.

The system has been classified into six clusters. The cluster at the top of Figure 5.25 includes the *sbse.handler* package, which is called by the external *Eclipse* plug-in framework to start the application. Additionally the *sourceReader* package is included,

which has a dependency to the *sbse.handler* package, but does not feature similar functionality.

The second cluster from the top exclusively comprises visualisation and tool packages. The main domain functionality of the SBRE is combined in the domain cluster. This comprises the *cluster*, *metric*, *representation* and *cluster.constraints* packages, which are known from the design and implementation of the SBRE software system.

The cluster below pools the source code reader and graph building functionality. Regarding the functionality of the included packages, the *sourceReader* package from the top cluster should have belonged in the *Source Reading* cluster. Additionally smaller clusters are identified, which contain library and base functionality. These clusters are visualized at the bottom of the graph.

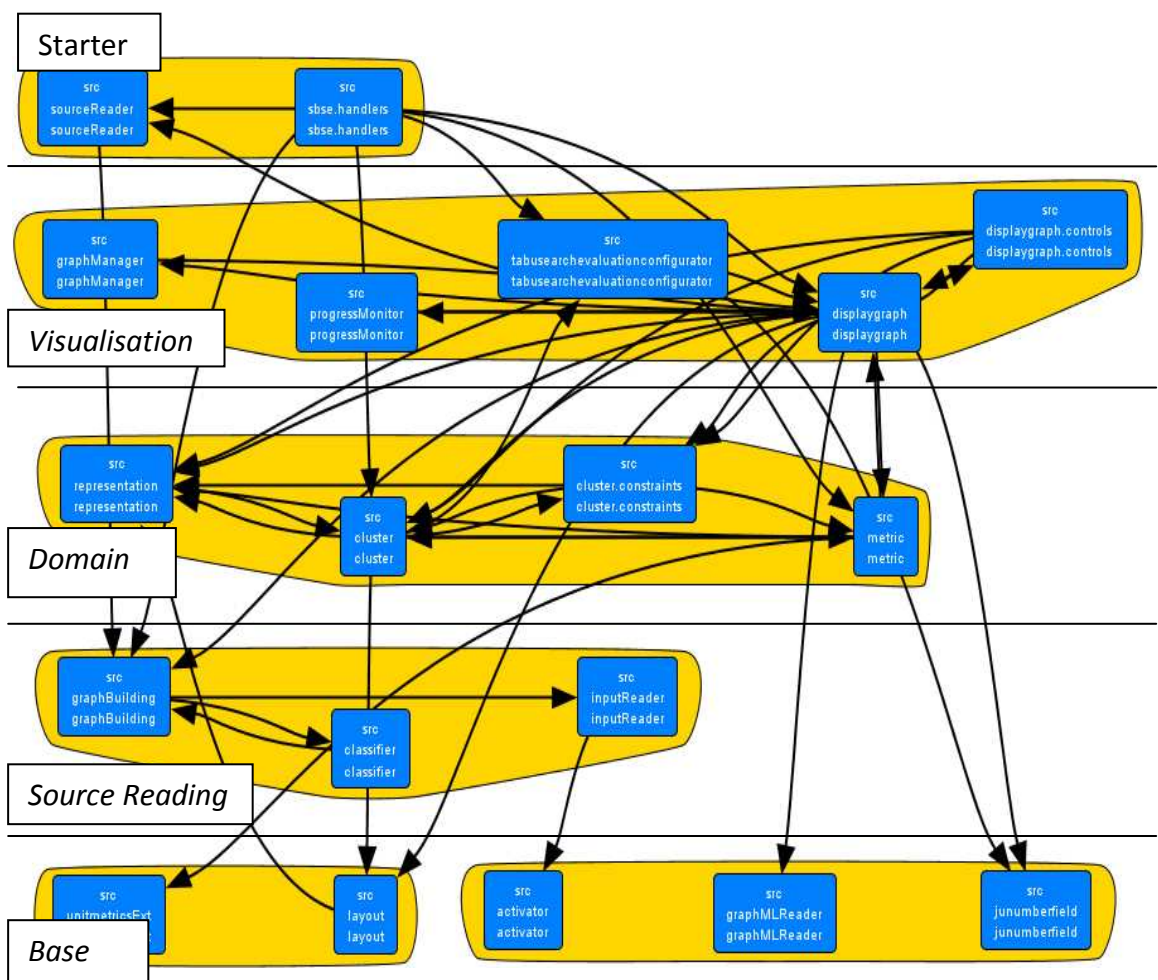


Figure 5.25 : Solution of the rearchitecting analysis of the SBRE component

It can be recognized, then, that the SBRE system is not perfectly designed. Assuming that the current cluster landscape reflects a sufficient modularisation of the system, some clusters show cycles between each other e.g. the *Domain* and *Visualisation* cluster (metric package -> displaygraph package -> metric package). These cycles reduce the independent reusability of the involved components and increase the complexity of the software system. Such cycles and architecture violations are a first lead for possible refactorings. Only the implementation of these refactorings would improve the quality of the software system. Additionally, it should be mentioned that these cycles and architecture violations misguide the cluster algorithm, as the cluster algorithms do not know if a dependency follows the normal dependency flow or if it is perhaps an architecture violation. Considering this, it is probably easier to derive an optimal structure from a clustered system when the system features minimal deterioration. This is also an argument to distribute the clustering on the measurement of different metrics and also consider if these measurements reflect the design of the system (as is done here in SBRE). The illustrated clustering result of the SBRE system with the focus on the *Cohesion Between Objects* and *Correlation of Names* did deliver a meaningful cluster landscape, which allowed the identification of the core modules of the system. Based on this, a first architecture definition can be derived – but as illustrated with examples this first solution can still be improved.

5.4.3 Inclusion of User Domain Knowledge

The main research question of this research is whether a user directed and semi-automatic clustering approach can contribute to the quality of software clustering. Different measures have been introduced within the SBRE component to enable the user to control the clustering process. The SBRE tool supports the definition of constraints for the manual refinement of solutions. The evaluation of this measure contributes to the evaluation of the main research question.

An experiment is conducted that evaluates the inclusion of user constraints in the cluster process. The baseline for this experiment is the cluster configuration and result from the previous section. As described in the previous section the *sourceReader*

package is combined with the *sbse.handler* package. However, from a functionality aspect it should be combined with the packages from the *Source Reading* cluster.

To force the search into this area of the search space, two constraints have been formulated. The first one disconnects the *sourceReader* package from the *sbse.handler* package and the second one connects the *sourceReader* package with the *graphBuilding* package. The search then allows only solutions, which fulfil these constraints. Figure 5.26 illustrates the new cluster landscape.

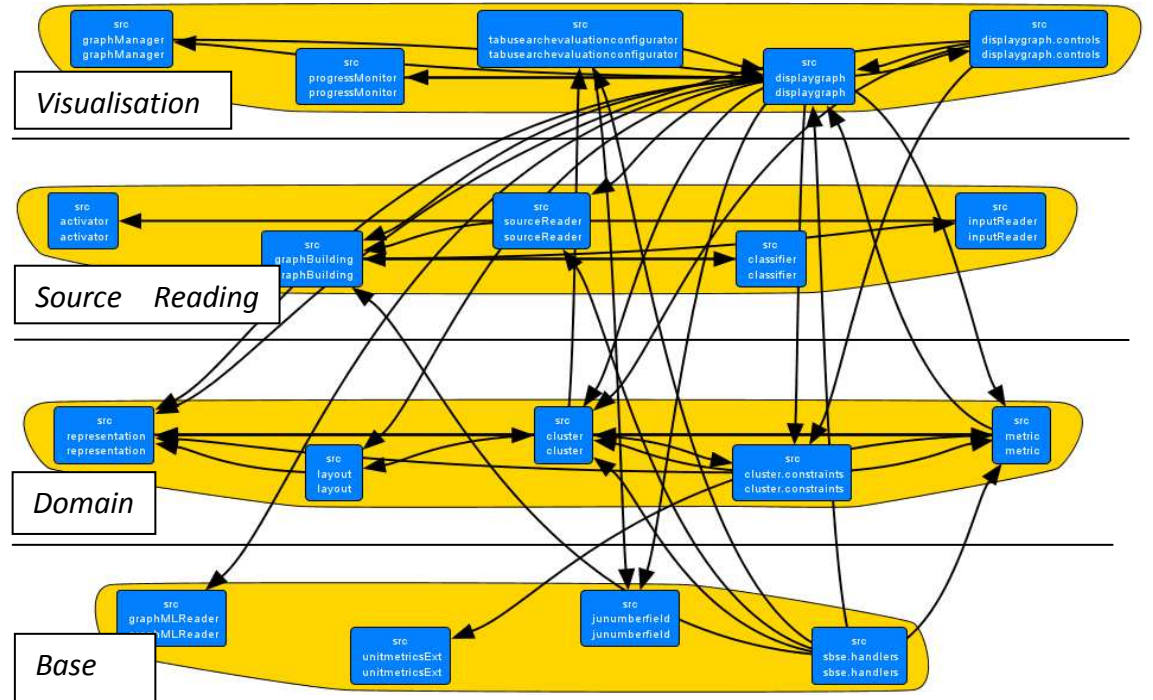


Figure 5.26 : Solution of the SBRE component with cluster constraints

The new cluster landscape consists of just five clusters. This solution aligns with the defined constraints. The main clusters *Visualisation*, *Source Reading* and *Domain* are fundamentally conserved. However, the inclusion of the constraints had also some side effects on packages, which were not included in the constraint definition. Examples of this are the allocation of the *activator* package into the *SourceReader* cluster and the classification of the *layout* package into the *Domain* cluster. This is plausible because, the results of the previous search are not influencing the search with the constraints. The new cluster landscape can also be a good solution – although it differs significantly from the previous one. This seemingly undirected behaviour may confuse the user. Furthermore, a strongly changed solution may discourage the user from trusting the

process. This apparently major change does not align with the idea of iterative user refinement of the solution, because the stakeholders would now have to understand and analyze the new solution and probably plan new constraints to correct the side effects of the last constraint inclusion. In turn, this may cause new side effects. The application of constraints in an interactive manner to refine a solution is therefore not effective as currently implemented in combination with metaheuristic algorithms. For a successful application of interactive solution refinement another path would have to be followed which aligns more with the results of the previous iteration.

5.4.4 Cluster Analysis on Different Abstraction Levels

Another user-directed clustering approach enabled within the SBRE component is the clustering of a software system on different abstraction levels. Specifically, the SBRE component currently provides the capability to cluster the system at the class or package level. This measure does provide more control over the software clustering process. Hence, the evaluation of this measure contributes to the evaluation of the main research objective, if a user directed and semi-automatic clustering approach can contribute to the quality of software clustering.

The previous experiments were applied at the package level, whereas this following experiment depicts clustering at class level. The same *TabuSearchStrategy* and metric configuration as portrayed in Figure 5.23 and Figure 5.24 are applied within this experiment. Accordingly, the result of the clustering, which is portrayed in Figure 5.27, is comparable with the clustering result of Figure 5.25.

The clustered SBRE system features 163 artefacts and 420 edges, with the clustering procedure combining these artefacts into 35 clusters. It has not been determined whether the created solution is meaningful in terms of the configuration. Rather, as intended, this experiment depicts the difference in complexity that results from the clustering of the same system at class and package level.

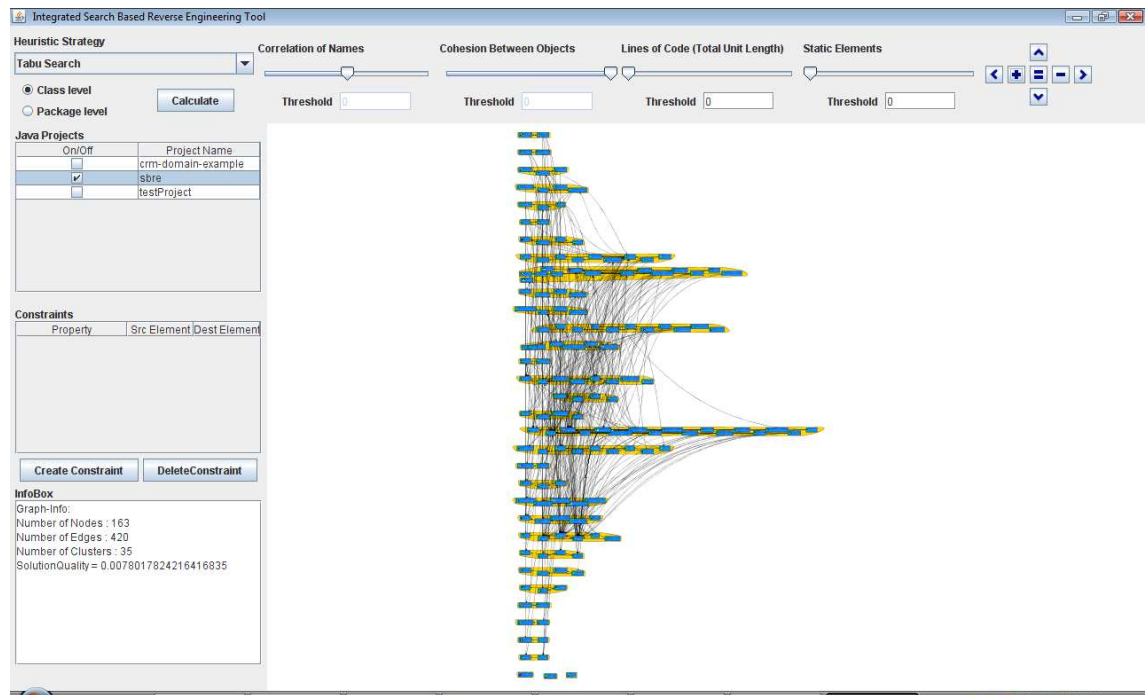


Figure 5.27 : Clustering of the SBRE system (class level)

The depiction of the identified solution at class level conveys a very high complexity in comparison to the software system analyzed at package level. In fact it would be challenging to recognize or derive a structure from the class-level clustered software system. The reason for this is the high number of clustered edges and artefacts within the graph, but also the overload of the visualization and navigation component. It remains to be investigated whether better concepts for visualization and navigation exist to illustrate and structure a software system of this size. This observation corresponds with that of Mitchell (2002), who stated that the clustering of a system with more than seventy-five artefacts tends to get confusing and that it is difficult to derive a structure from such a system. In conclusion, the clustering at higher abstraction levels can hide the complexity of software systems and enables the derivation of a suitable structure (at that level) for larger software systems. For even larger systems, it may be necessary to consider yet higher levels of abstraction. For these systems it may be difficult to derive a structure at the package level, so subsystems may need to be used.

5.5 Comparison of the SBRE component with *Bunch* and *Barrio*

The *Barrio* and *Bunch* clustering components had a substantial influence to the present research. This section emphasizes the differences of these clustering components in comparison with the SBRE component. This comparison contributes to the evaluation of the main research question, as the comparison with the *Barrio* and *Bunch* component delivers a benchmark for the evaluation, if a user directed and semi-automatic clustering approach is able to contribute to the quality of software clustering.

The comparison with the *Barrio* and *Bunch* framework is not a complete review of the functionality of these tools. Only functionality is discussed, which relates to the objectives of this research. The comparison of specific clustering results is not conducted, as all three components offer the possibility to influence the clustering output and as the parameterization follows different aspects the results are not comparable. Additionally, a comparison of specific cluster results would, in the opinion of the researcher, not contribute to the objectives of the present research. Within this section it is rather interesting, which functionality is offered by the SBRE component in comparison to the functionality of the *Bunch* and *Barrio* component.

Barrio Component

The *Barrio* component delivered the source code to enable an early applicability of the SBRE component. It contributed the “exposure of the artefacts” mechanism, the navigation component and partially the visualisation component. However, the *Barrio* component follows no search based cluster strategy and offers only a limited possibility for the user to influence the clustering. The only measure for the stakeholder is the separation level, which defines the minimum number of dependencies between two sub graphs. The separation level is certainly of importance for special analysis such as the identification of starting points for the decoupling of sub graphs. However, the measurement itself comprises also a certain degree of subjectivity as no other clustering criteria can be determined. In comparison, the SBRE component enabled the minimisation of this subjectivity by applying a multi metric approach. The *Barrio* component does not feature the inclusion of user constraints or

the clustering on different abstraction levels. A feature of the *Barrio* component is that the clustering result can be compared with the package structure of the system. Finally, no artefact ordering is suggested from the *Barrio* component, this reduces the clearness of a cluster landscape in comparison to the visualisation of the SBRE component, which offers an initial ordering of the cluster landscape.

Bunch Component

The *Bunch* component as a SBSE based clustering component features a similar cluster strategy as the SBRE component. With the difference that the SBRE component applies different metaheuristics (greedy and tabu search) and focuses on the examination of a multiple metric and user directed clustering process.

The *Bunch* component, with the application of *Genetic Algorithms*, does not follow a deterministic clustering approach. As a consequence the cluster results differ with each attempt at the clustering. In comparison, the SBRE component utilises a deterministic approach. The non-deterministic approach of the *Bunch* component is confusing for the stakeholder as the user has to trigger the clustering repeatedly until a good solution is found.

To conserve the extensibility of the visualisation no data exchange between the visualisation component and the *Bunch* component is designated. The *Bunch* component utilizes the external graph visualisation component *dotty*².

The cluster process of the *Bunch* component is completed with the visualisation of the cluster landscape. This prohibits any direct user interaction with the visualized cluster landscape. In comparison, the *SBRE* component features the functionality to arrange artefacts, add user-constraints, use the current cluster result as an initial solution for the next clustering, configure thresholds, exclude artefacts and change the metric configuration. This interaction with the visualisation component enables the iterative and incremental clustering approach of the SBRE component. The disadvantage is a

² <http://graphviz.org/>

strong interconnection and dependency between the SBRE clustering and visualisation component, which prohibits the individual employment of the SBRE clustering process.

In retrospect, and with regards to the reduction of the metric measurements into one compressed dependency similarity within the *SBREMetricEngine*, the design of the *SBRERepresentation*, the design of the *TabuSearchStrategy* and the applicability of the *CDA* component, it seems to be promising that the examination of the multiple metric approach to create different implementation perspectives and the examination of the applicability of tabu search would have been also possible by applying these approaches in combination with the *Bunch* component. However, this solution would have prohibited an interactive, integrated and user directed approach. This means that, the lack of interface and communication mechanisms within the *Bunch* component between the exposure of the artefact, the clustering process and the visualisation component would have prohibited the adjustment of the metric weighting and tabu search configuration within a *Bunch* based solution.

In conclusion, it can be stated that the *Bunch* component as a SBSE based clustering component follows a similar clustering approach as the SBRE component. However, the non-deterministic character of the *GA* algorithm of the *Bunch* component introduces additional complexity for the stakeholder to select the optimal solution, as well as making a comparison of results across different tools more complex. Additionally, the detachment of the *Bunch* clustering component and the visualisation prohibits the interactive manipulation of the cluster landscape. It can be stated that the interactive, iterative and incremental clustering approach of the SBRE component contributed to enable a flexible user-directed software clustering environment. This allows the continuing of the clustering process, where the *Barrio* and *Bunch* component stopped their cluster analysis.

5.6 Summary

This chapter evaluated the capability of the SBRE component to cluster a software system into subsystems. Different aspects of the SBRE component have been examined. The *TabuSearchStrategy* and *GreedyBestNeighbour* algorithm are evaluated. A runtime evaluation of both algorithms and comparison of the runtime behaviour has been conducted. Furthermore, an evaluation of the *TabuSearchStrategy* parameters is implemented to examine the applicability of the tabu search concepts in the area of software clustering. Furthermore, the capability to adapt the clustering result to the metric configuration has been evaluated. It has been shown that the SBRE component is able to indentify interrelated clusters, which feature a certain degree of deterioration. Additionally, it has been shown that the SBRE system is able to identify the core components of the SBRE system in a special configuration of metrics. It has been illustrated that this modularisation can be used as a first draft to define an architecture definition. However, it has also been acknowledged that the SBRE approach will not be able to derive a complete target-architecture for a system as the cluster algorithms cannot distinguish between dependencies that follow the normal dependency flow or are architecture violations. The latter have the potential to misguide the clustering process. In addition, the constraint mechanism of the SBRE system has been applied to refine an existing cluster solution. It has been shown that this may result in side effects and lack accordance with a previous solution, which could confuse the user and therefore disqualifies the application of interactive constraints in combination with metaheuristic algorithms. Finally, a comparison of the SBRE clustering approach with the *Barrio* and *Bunch* component has been conducted.

6 Limitations, Future Research and Conclusion

The following chapter answers the identified research questions, illustrates limitations, discusses suggestions for further research and draws a final conclusion for the conducted research.

6.1 Answer of the Research Questions

This research has applied SBSE in the area of software clustering. Based on a literature review in the areas software-architecture, SBSE, software-clustering and software metrics the research objective has been derived. Regarding the constructive and explorative character of this research project the SDRM was applied. The following three research questions have been identified and answered during this research.

- Can a user directed and semi-automatic clustering approach contribute to the quality of software clustering?
- Is tabu search applicable in the area of software clustering?
- Does the inclusion of multiple metrics in the fitness function enable the clustering of a software system into multiple implementation perspectives?

To enable the examination of the research objective the SBRE component has been designed and implemented. The SBRE component allowed formulated research questions to be addressed by conducting a range of experiments that involved the clustering of two software products. Based on this experimentation the findings can be discussed and the answers to the research questions can be addressed. The ordering of the research questions is changed for the answering, as the answer of the second and third research question also contributes to the answer of the main research question.

6.1.1 Application of the Tabu Search Algorithm

This subsection discusses the findings regarding the research question: 'Is tabu search applicable in the area of software clustering?'.

Two algorithms have been designed, implemented and evaluated within this research, namely the *GreedyBestNeighbour* and the *TabuSearchStrategy* algorithm. These

algorithms are specific implementations of the greedy and tabu search metaheuristics respectively, and their development was informed by consideration of the research literature in this area.

In section 5.4.2 it has been shown that clustering with the *TabuSearchStrategy* allows the identification of a structure of the SBRE software system. In section 5.3.1 and 5.3.2 a performance analysis of the *GreedyBestNeighbour* and the *TabuSearchStrategy* is described. The results of these analyses are compared with each other. It has been shown that the *TabuSearchStrategy* algorithm is able to effectuate an improvement of the *SolutionQuality* in comparison to the best identified solution of the *GreedyBestNeighbour* algorithm. Whilst the *TabuSearchStrategy* demands more computational resources to achieve this improvement in *SolutionQuality*, the difference is insignificant when put into the context of actual use by a human software developer. It has been found that varying the length of the tabu list can prevent confinement in cycles within the SBRE tool and as a consequence enables the improvement of the *SolutionQuality*.

Furthermore, the triggering of diversification has an influence on the search process and is able to affect the cluster process. It has not been investigated whether the applied intensification and diversification methods are optimal or if other approaches would lead to a better solution outcome. This would be useful future work. In this respect it should be noted that the actual effect of intensification and diversification on the search result would also depend on the analyzed software system and the related parameters. As such, the application of the intensification and diversification can be beneficial but also could negatively impact the results of the search.

In conclusion, the *TabuSearchAlgorithm* as a specific implementation following the concepts of Glover (1989) is applicable in the area of software clustering. Additionally, the *TabuSearchAlgorithm* features a variety of parameters. The configuration of these parameters enables the guiding of the search and the adaption to the preferences of the stakeholder. This flexibility allows the user to adjust parameters and return to previous solutions. However, the complexity of the algorithm requires that the user has a good understanding of the tabu search concept in order to apply and configure

the algorithm efficiently. At present, there is a limited body of knowledge in the area of search based clustering, primarily formed by the work of Mitchell (2002), Jiang et al. (2007) and Seng et al. (2005). This research contributes to this body of knowledge with the development of *GreedyBestNeighbour* and *TabuSearchAlgorithm*, both of which are deterministic search based clustering approaches. The deterministic nature of both algorithms increases the transparency for the user and enables them to return to previous solutions given a particular parameter configuration.

6.1.2 Multiple Implementation Perspectives

The following subsection discusses the findings regarding the second research question: ‘Does the inclusion of multiple metrics into the fitness function enable the clustering of a software system into multiple implementation perspectives?’

To examine this research objective the four metrics *Correlation of Names*, *Cohesion Between Objects*, *Total Lines of Code* and *Number of Static Elements* have been implemented. The user can individually control the weighting of each metric which defines the degree of consideration given to each metric during the clustering process. In essence, this weighting of metrics gives the user the ability to control the nature of the fitness function to ensure that the clustering process is directed towards solutions that suit a particular set of preferences or project constraints. This is achieved because the metric measurements and their weights are considered by the fitness function when evaluating a solution. To examine the effectiveness of this approach, experiments have been conducted that assess whether the metric configuration is able to guide the search into areas of the search space that align with the metric configuration and hence the users preferences.

In alignment with the research of Harman & Clark (2004) it has been shown that metrics are able to influence the clustering of software systems in an understandable and consistent way. During the evaluation it has been found that the multiple-metric approach is able to produce different cluster landscapes, which align with the metric configuration and the users preferences. This enables the stakeholder to guide the search into the intended areas of the search space. These intentions can exhibit different characteristics; examples would be to identify highly cohesive clusters within

the system or to isolate clusters that feature high name similarity of the included artefacts. It has also been demonstrated that the metric configuration can be aligned with the design strength of the analyzed software system. This simplifies the identification of feasible solutions for the cluster algorithms. For example, in some cluster analyses, the search can be guided to more feasible solutions if the fitness function focuses on the *Correlation of Names* metric. Other software systems feature a good modularity. In this case the system can be more effectively clustered by applying the *CohesionBetweenObjects* metric. A combination of metrics can also deliver an optimal solution, if the analyzed software system design reflects this combination. This optimal solution is balanced by the multi-metric approach so whilst the solution exhibits certain characteristics, these characteristics do not dominate the solution by taking them to extremes.

It has also been shown that a multiple-metric approach is able to identify a cluster landscape that features components with high functional cohesion. This cluster landscape could be effective in enabling the user to improve their understanding of the software system. This system understanding, in association with the domain and system knowledge of the stakeholders, could provide a sound basis for an architecture definition. As such, this work is in agreement with the research of Mitchell (2002) in stating that a software clustering approach can contribute to an increase in system understanding and, as a consequence, deliver a basis for the derivation of an architecture definition. It is therefore not in agreement with the research of Anquetil & Lethbridge (1999b), which states that a software clustering approach is able to derive an architecture definition from a system, one that can then be used as an architecture pattern for the further development process. The reason for this is that even if a global optimum is identified during the search, this optimum may not necessarily align with the expected result of the stakeholder. This global optimum represents only the optimum according to the selected fitness function. It is doubtful whether a specific metric configuration and a selected fitness function respectively will be able to optimize the intended architecture of a system. Given the additional assumption that every software system features a certain degree of deterioration, it is even more likely that the global optimum will not represent the best solution for the stakeholder, as the

deterioration will guide the search into areas of the search space that do not align with the intended architecture design. Additionally, it has been evaluated that it is possible to identify clusters that feature a high occurrence of threshold violations. This enables the user to identify “code smells” on an abstract level. These “*code smell*” clusters can be considered for the planning of refactorings.

In conclusion, it can be stated that a flexible multiple metric approach supports the adaptation of the search to the preferences of the stakeholders and to the design and characteristics of the analyzed software system within the implementation perspective. But the success of a metric configuration to gather the intended structure of the software system depends also on the quality of the design of the software system.

6.1.3 User Directed Software Clustering

This subsection discusses the findings regarding the main research question ‘Can a user directed and semi-automatic clustering approach contribute to the quality of software clustering?’.

Within section 3.2 it has been emphasised that the term ‘quality of software clustering’ within this work focuses on the feasibility of the identified solution and the flexibility of the clustering process. The SBRE component features four mechanisms to enable the user to control the clustering process.

The first is the adjustment of metric weights to guide the search into the intended areas of the search space. The applicability of this approach is discussed in the previous section.

It has been shown in section 5.4.1, that the configuration of the metric configuration allows the clustering result to be aligned with the preferences and requirements of the stakeholders. This approach increases the feasibility of the solution in relation to the requirements of the stakeholder. This feasibility is based primarily on the minimization of subjectivity and the elimination of dependence on just one fixed measurement when identifying a cluster solution, and hence a more balanced solution can be determined.

The second user-directed mechanism is the capability for the user to adjust parameters of the *TabuSearchStrategy* algorithm, which enables the user to adapt the search process to the requirements of the search and the analyzed software system. Clearly not all software systems are similar, depending on their size and structure different parameters will guide the search to better results. It has been shown in section 5.4.1 that the configuration of the *TabuSearchStrategy* algorithm is able to find different clustering results, depending on the algorithm configuration. Regarding this, the opportunity to configure these parameters increases the flexibility of the clustering approach and affects the search. However, the necessity to configure the parameters of the *TabuSearchStrategy* algorithm increases the complexity of the clustering in comparison to the application of the *GreedyBestNeighbour* algorithm and forces the user to have a good understanding of the tabu search concepts. Additionally, the transparency and effect of the intensification and diversification triggering is not clear for the user, as the success of the triggering depends also on the analyzed software system and the configuration of the other tabu search parameters.

The third mechanism is the facility to include user constraints. This mechanism enables the user to include domain knowledge into the clustering process. It has been shown in section 5.4.3 that user constraints are applicable to force the metaheuristic algorithms into certain areas of the search space. One could say that already the fulfilment of this technical requirement increases the quality of the clustering result as the result features the user constraint. However, it has also been shown that the inclusion of user constraints influences the clustering of the remaining artefacts. These side effects increase the complexity imposed on the user in terms of their understanding of the new clustering result and destroy the recognition value from the previous solution.

Finally, the last user directed approach examined the clustering on different abstraction levels. Except from the illustration of the clustering on class level in section 5.4.4 and the identification of deteriorated subsystems in section 5.4.2, every experiment is conducted on package level. The experiment in section 5.4.4 shows that the clustering of the SBRE system on class level features a high degree of complexity. Hence it is hard to identify system structures. The clustering on package level within

bigger systems contributes to the understandability of the cluster landscape and simplifies the derivation of a structure. The novelty of this approach is the contribution of the multi-metric approach to the clustering on different abstraction levels. The metric values are compressed from the *class* into the *package* level depending on the metric type. Regarding this, the approach does not only reduce the amount of artefacts, it also incorporates the measured metric values and therefore also enables the clustering on package level with the inclusion of a multi metric approach.

The clustering of smaller software systems on class level could be beneficial, even if it is questionable if the need exists to cluster small software systems. Furthermore, for the derivation of deteriorated subsystems, the clustering on class level is beneficial. Here the number of artefacts to be clustered is reduced by the inclusion of thresholds.

In light of the results presented in this research, no firm conclusion about whether a user directed approach increases the quality of software clustering can be made. Each aspect used in the clustering process can only be evaluated individually in terms of how it influences the end result. Additionally, the developed mechanisms are only examples for the possibilities of user directed approaches. Certainly, further user directed approaches are imaginable to give the user more control over the clustering process.

But it can be stated that the application of certain user directed clustering approaches has the potential to increase the quality of the clustering results in comparison to other zero- or single-measurement-based approaches. Further work is necessary to quantify this improvement across a broader range of experiments. However, it should be noted that the flexibility gain is accompanied by an associated increase in the complexity of the clustering configuration. The user requires a good knowledge about the domain and architecture of the software to effectively use the described functionality to guide the search process.

Additionally, to effectively manage the application of specialised algorithms, e.g. the *TabuSearchStrategy* algorithm, a high level of understanding of the method of operation is required. Furthermore, it remains the position of this research that the

variety of solutions and requirements within the area of software clustering are so complex that a fully automated approach would lead to unsatisfactory results and cannot fulfil all user requirements. In contrast, a manual approach requires many decisions to be taken, which increases the complexity of the clustering. Hence, it can be stated that the flexible combination of semi-automated or user-directed clustering and the capacity for user adjustment represents the most promising approach. It enables the user to decide if a quick solution is preferred, without the necessity of user configuration, or if a more specialized solution is needed, which requires the inclusion of domain-knowledge and the alignment of the available metrics.

6.2 Limitations

This section describes the limitations of the research that has been conducted. This research project is applied in an experimental environment. The evaluation of the developed approach is only conducted with a small set of test data, which was mainly the SBRE system itself. This has shown the applicability of a user directed SBRE based approach in the area of reverse engineering, but does not feature sufficient scale or variety to allow a general statement to confirm if this approach is also feasible for other software systems, which exhibit a different structure and size. Furthermore, the SBRE system is restricted to the analysis of Java systems. This restriction is based on the application of the *Eclipse* environment and the application of software metrics which are only able to analyze *Java* code.

As described in section 4.2, the SBRE component applies the *CDA* component to extract the dependency graph of the analyzed software system. A substantial part of current software frameworks(e.g. *EJB (3.0+)*, *Spring*, *PicoContainer* for *Java* systems or *Spring.Net*) feature a concept called dependency injection. Dependency Injection is a design pattern, which allows reducing the dependencies between components and objects in object oriented systems. Dependency Injection is an application of the *Inversion of Control (IoC)* principle and can be understood as a generalization of the factory method approach. Dependency Injection assigns the responsibility for the creation and the linkage from objects to an external framework. Through this assignment the source code is removed from its environment and from the specific

implementation of the classes. The *CDA* component does not consider dependencies which are injected by inversion of control frameworks. Regarding this the *SBRE* component cannot analyze software systems that feature dependency injection.

6.3 Future Directions

This section describes some suggestions for the extension of this research project and opportunities for future research directions are indicated.

This work illustrated the applicability of the tabu search algorithm in the area of software clustering. It has been examined that the tabu search concepts and parameters are influencing the solution finding process within the area of software clustering. The main tabu search concepts (e.g. tabu list concepts, intensification and diversification) have been implemented in a very basic manner. The results of the intensification and diversification approach were poor within the conducted experiments. It would be a point of further research to examine if these approaches are feasible or if better concepts are imaginable. Furthermore, it would be interesting if a correlation exist between an optimal tabu search parameter configuration and the characteristic of the analyzed software system.

It could also be considered to apply statistical and random approaches, which would break the deterministic character of this research, but also have the potential to discover better solutions.

It has been shown within this research that it is possible to create different implementation perspectives for a software system by applying software metrics. The chosen metrics represent only a very small compendium of the available metrics. Especially, the approach to include metrics that measure attributes of the artefact itself opens a variety of possibilities for the application of other metrics. It would be interesting to further investigate the capability and benefits of the identification of abstract refactoring units. However, it should be recognised that including additional metrics does increase the complexity of choosing how each metric should be applied.

As shown within the present research the developed fitness function features some limitations, which does not allow exploiting the full capacity of a user directed

clustering approach. The aim within this research was to implement a fitness function which selects feasible solutions and supports the examination of the research objective. As described, the fitness function does not consider the similarity of artefacts which do not exhibit a dependency between each other. This anticipates the clustering of artefacts which do not feature an interrelation between each other. It would be interesting to investigate if the clustering of non-dependant artefacts allows deriving clustering landscapes which provide meaningful cluster information for the stakeholders. Another qualification of the fitness function is the penalization of small and big clusters. During the design and implementation phase the fitness function selected solutions which exhibit a high degree of homogeneity. These solutions were hard to understand and infeasible for the analyses of software systems. Based on this, a penalizing strategy was introduced to create more homogeneous solutions. This approach improved the solution finding process and created solutions, which were easier to understand. But especially, if the design of the analyzed software system does not align with the aim of the metric configuration, this penalizing strategy forces the search into areas of the search space which do not reflect the aim of the metric configuration. It would be interesting to examine if a more flexible configuration of the fitness function would allow better adaptation of the search to the requirements of the stakeholders. Finally, the evaluation has been applied in a very small scope. The next step would be to apply the SBRE approach within larger environments and conduct software cluster analysis outside of these experimental conditions.

6.4 Conclusion

At the beginning of this research has been stated that within the past forty years no approach has arisen to confine the *software crisis*. Therefore the problem still exists as to how to control and maintain non trivial software systems. Certainly, the presented approach is not the only possible solution to confine the software crisis, or indeed solve the software crisis in its entirety. But it has been shown that the proposed approach has the potential to help development stakeholders to create abstract perspectives of the system structure. This is the key contribution of this work to the body of knowledge in this field. The SBSE based clustering approach can help these stakeholders to gather information about the software system, which can be utilized for the further development, design and maintenance of the system. This increase of understanding of software systems can contribute to the confinement of software deterioration and therefore potentially enhance the efficiency of the software development and maintenance processes.

The novelty of the presented approach is the focus on the user directed clustering. This approach allows the alignment of the clustering with the requirements of the stakeholders and the design of the software system. A variety of user directed measures have been introduced and evaluated within this research to enable the stakeholder to influence the clustering process without being occupied with trivial clustering tasks. It has been found that user directed clustering approaches have the potential to contribute to the quality of software clustering in comparison with other non-interactive and non-user directed approaches. As shown during the present research this increases the application area of the clustering approach and removes subjectivity of the clustering process. Finally, it can be stated that this research illustrated the feasibility of a user directed and interactive SBSE based clustering approach and reported on the benefits for development stakeholders. The next step would be the enlargement of the conducted research to evaluate the applicability outside of an experimental environment.

7 References

- Anquetil, N., & Lethbridge, T. C. (1999a). *Experiments with clustering as a software remodularization method*. Paper presented at the Sixth Working Conference on Reverse Engineering (WCRE), Atlanta, Georgia, USA 6-8 October 1999.
- Anquetil, N., & Lethbridge, T. C. (1999b). Recovering software architecture from the names of source files. *Journal of Software Maintenance: Research and Practice*, 11(3), 201-221.
- Bäck, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford: Oxford University Press.
- Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of evolutionary computation*. New York: Taylor & Francis Group.
- Banker, R. D., Datar, S. M., Kemerer, C. F., & Zweig, D. (1993). Software complexity and maintenance costs. *Communications of the ACM*, 36(11), 81-94.
- Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Reading, Massachusetts: Addison-Wesley Professional.
- Birattari, M. (2005). *The problem of tuning metaheuristics: As seen from a machine learning perspective*. Amsterdam: IOS Press Publication.
- Bischofberger, W., Kuhl, J., & Löffler, S. (2004). Sotograph - A pragmatic approach to source code architecture conformance checking. In G. Goos, J. Hartmanis & J. Van Leeuwe (Eds.), *Software architecture* (pp. 1-9). Berlin: Springer.
- Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., & Houston, K. (1994). *Object-oriented analysis and design with applications*. Reading, Massachusetts: Addison-Wesley Professional.
- Bosch, J. (2004). Software architecture: The next step. In G. Goos, J. Hartmanis & J. Van Leeuwe (Eds.), *Software architecture* (pp. 194-199). Berlin: Springer.
- Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6), 476-493.
- Chikofsky, E. J., & Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *Software IEEE*, 7(1), 13-17.
- Collis, J., & Hussey, R. (2009). *Business research: A practical guide for undergraduate and postgraduate students* (3rd ed.). Basingstoke: Palgrave Macmillan.
- Connor, A., Clarkson, P. J., Shahpar, S., & Leonard, P. (2000, 27-29th June). *Engineering design optimization using Tabu search*. Paper presented at the Proceedings of Design for Excellence: Engineering Design Conference (EDC 2000), Brunel, UK.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). *Introduction to algorithms* (2nd ed.). Massachusetts: The MIT Press.
- Counsell, S., Swift, S., & Crampton, J. (2006). The interpretation and utility of three cohesion metrics for object-oriented design. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2), 123-149.
- Courses, E., & Surveys, T. (2002). *Object-oriented system decomposition quality*. Paper presented at the High Assurance Systems Engineering, 2002. 7th IEEE International Symposium on 23-25 October 2002, Tokyo, Japan.

- Darcy, D. P., & Kemerer, C. F. (2002). *Software complexity: Toward a unified theory of coupling and cohesion*. Paper presented at the International Conference on Software Engineering (ICSE), Orlando, Florida, USA 19-25 May 2002.
- DeVore, R. A., & Temlyakov, V. N. (1996). Some remarks on greedy algorithms. *Advances in Computational Mathematics*, 5(1), 173-187.
- Dietrich, J. (2009). Barrio Project Home Page. Retrieved 15/09/2009, from <http://code.google.com/p/barrio/>
- Dietrich, J., Yakovlev, V., McCartin, C., Jenson, G., & Duchrow, M. (2008). *Cluster analysis of Java dependency graphs*. Paper presented at the SOFTVIS 2008: 4th ACM Symposium on Software Visualization, Herrsching am Ammersee, Germany, September 16-17, 2008.
- Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Massachusetts: The MIT Press.
- Duchrow, M. (2009). Class dependency analyzer Retrieved 30/07/2009, from <http://www.dependency-analyzer.org/>
- El-Wakil, M., El-Bastawisi, A., & Boshra, M. (2004). Software metrics - A taxonomy. Retrieved 10/05/2008, from http://homepages.wmich.edu/~m5elwakil/CSITeA04_ElWakil.pdf
- Erhard, K. (1991). Software metrics, measurement theory, and viewpoints. *SIGPLAN Notices*, 26(3), 53-62.
- Fenton, N., & Pfleeger, S. L. (1997). *Software metrics: A rigorous and practical approach*. Boston, Massachusetts, USA: PWS Publishing Co.
- Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Reading, Massachusetts: Addison-Wesley Professional.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, Massachusetts: Addison-Wesley Publishing Co.
- Girvan, M., & Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12), 7821.
- Glass, R. L. (2002). Sorting out software complexity. *Communications of the ACM*, 45(11), 19-21.
- Glover, F. (1989). Tabu search--part I. *INFORMS Journal on Computing*, 1(3), 190.
- Glover, F. (1990). Tabu search-part II. *ORSA Journal on computing*, 2(1), 4-32.
- Glover, F. (1995). *Tabu search fundamentals and uses*. Boulder, Colorado: Graduate School of Business, University of Colorado
- Glover, F., & Kochenberger, G. A. (2003). *Handbook of metaheuristics*. Berlin: Springer.
- Glover, F., & Laguna, M. (1997). *Tabu search*. Berlin: Springer.
- Gui, G., & Scott, P. D. (2006). *Coupling and cohesion measures for evaluation of component reusability*. Paper presented at the MSR 2006: International Workshop on Mining Software Repositories 22-23 May 2006, Shanghai, China.
- Harman, M., & Clark, J. (2004). *Metrics are fitness functions too*. Paper presented at the International Software Metrics Symposium, Chicago, Illinois, USA, 14-16 September 2004.
- Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and Software Technology*, 43(14), 833-839.

- Heer, J., Card, S. K., & Landay, J. A. (2005). *Prefuse: A toolkit for interactive information visualization*. Paper presented at the SIGCHI Conference on Human Factors in Computing Systems, Portland, Oregon, USA 2-7 April 2005.
- Hitz, M., & Montazeri, B. (1996). Chidamber and Kemerer's metrics suite: A measurement theory perspective. *IEEE Transactions on Software Engineering*, 22(4), 267-271.
- Hofmeister, C., Nord, R., & Soni, D. (2000). *Applied software architecture: A practical guide for software designers*. Reading, Massachusetts: Addison-Wesley Professional.
- Hunold, S., Krellner, B., Rauber, T., Reichel, T., & Rünger, G. (2009). *Pattern-based refactoring of legacy software systems*. Paper presented at the ICEIS International Conference on Enterprise Information Systems, Milan, Italy 6-10 May 2009.
- Jiang, T., Gold, N., Harman, M., & Li, Z. (2007). Locating dependence structures using search-based slicing. *Information and Software Technology*, 50(12), 1189-1209.
- Jones, M. R. (2004). Debatable advice and inconsistent evidence: Methodology in information systems research. In B. Kaplan, D. P. Truex III, D. Wastell, A. T. Wood-Harper & J. I. DeGross (Eds.), *Information Systems Research: Relevant Theory and Informed Practice* (pp. 121-142). Boston: Springer.
- Kennedy, J., Eberhart, R. C., & Shi, Y. (2001). Swarm intelligence. In A. Y. Zomaya (Ed.), *Handbook of nature-inspired and innovative computing integrating classical models with emerging technologies* (pp. 187-219). Boston: Springer.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671-680.
- Knox, S. T. (1993). Modeling the cost of software quality. *Digital Technical Journal*, 5(4), 9-17.
- Lee, H., & Yoo, C. (2000). A form driven object-oriented reverse engineering methodology. *Information Systems*, 25(3), 235-259.
- Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2), 498-516.
- Mancoridis, S., Mitchell, B. S., Chen, Y., & Gansner, E. R. (1999). *Bunch: A clustering tool for the recovery and maintenance of software system structures*. Paper presented at the IEEE International Conference on Software Maintenance (ICSM'99), Oxford, England, UK 30 August - 3 September 1999.
- Mancoridis, S., Mitchell, B. S., Rorres, C., Chen, Y., & Gansner, E. R. (1998). *Using automatic clustering to produce high-level system organizations of source code*. Paper presented at the 6th International Workshop on Program Comprehension (IWPC'98) Ischia, Italy 24-26 June 1998.
- Martin, R. (1994). *OO Design Quality Metrics-An Analysis of Dependencies Report On Object Analysis and Design 2*.
- Mens, T., & Demeyer, S. (2001). *Future trends in software evolution metrics*. Paper presented at the International Workshop on Principles of Software Evolution, Vienna, Austria 10 - 11 September 2001.
- Mitchell. (2002). *A heuristic search approach to solving the software clustering problem*. Drexel University, Drexel.

- Mitchell, & Mancoridis. (2001). *Comparing the decompositions produced by software clustering algorithms using similarity measurements*. Paper presented at the International Conference on Software Maintenance (ICSM 2001), Florence, Italy 6-10 November 2001.
- Mitchell, & Mancoridis. (2006). On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering*, 32(3), 193-208.
- Mitchell, & Mancoridis. (2008). On the evaluation of the Bunch search-based software modularization algorithm. *Soft Computing- A Fusion of Foundations, Methodologies and Applications*, 12(1), 77-93.
- Mitchell, B. S., & Mancoridis, S. (2001). *Craft: A framework for evaluating software clustering results in the absence of benchmark decompositions*.
- Muller, H. A., Orgun, M. A., Tilley, S. R., & Uhl, J. S. (1993). A reverse engineering approach to subsystem structure identification. *Practice*, 5(4), 181-204.
- Nunamaker, J. F., & Chen, M. (1990). *Systems development in information systems research*. Paper presented at the Twenty-Third Annual Hawaii International Conference on System Sciences, Kailua-Kona, HI, USA 2-5 January 1990.
- O'Madadhain, J., Fisher, D., & Nelson, T. (2009). Java Universal Network/Graph Framework. Retrieved 15/09/2009, from <http://jung.sourceforge.net/index.html>
- Ott, L. M., & Thuss, J. J. (1993). *Slice based metrics for estimating cohesion*. Paper presented at the First International Software Metrics Symposium 1993, Baltimore, MD, USA 21-22 May 1993
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053-1058.
- Pham, H. (2003). Software reliability and cost models: Perspectives, comparison, and practice. *European Journal of Operational Research*, 149(3), 475-489.
- Randell, B. (1996). *The 1968/69 NATO software engineering reports*. Paper presented at the Dagstuhl Seminar on: The History of Software Engineering, Schloss Dagstuhl, Leibniz, Germany 26 - 30 August 1996.
- Riel, A. J. (1996). *Object-oriented design heuristics*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. .
- Rosenberg, J. (1997). *Some misconceptions about lines of code*. Paper presented at the 4th International Symposium on Software Metrics, Albuquerque, NM, USA 5-7 November 1997.
- Salger, F., Bennicke, M., Engels, G., & Lewerentz, C. (2008). *Comprehensive architecture evaluation and management in large software-systems*. Paper presented at the Quality of Software Architecture Conference (QoSA 2008), University of Karlsruhe (TH), Germany 14-17 October 2008.
- Schneidewind, N. F. (1992). Methodology for validating software metrics. *IEEE Transactions on Software Engineering*, 18(5), 410-422.
- Schwanke, R. W., & Hanson, S. J. (1994). Using neural networks to modularize software. *Machine Learning*, 15(2), 137-168.
- Seng, O., Bauer, M., Biehl, M., & Pache, G. (2005). *Search-based improvement of subsystem decompositions*. Paper presented at the Genetic and Evolutionary

- Computation Conference (GECCO 2005), Washington D.C., USA 25-29 June 2005.
- Shaw, M., & Garlan, D. (1996). *Software architecture*. Upper Saddle River, N.J.: Prentice Hall.
- Smith, C. U., & Williams, L. G. (2000). *Software performance antipatterns*. Paper presented at the 2nd International Workshop on Software and Performance (WOSP 2000), Ottawa, Canada 17-20 September 2000.
- Sung, C. S., & Jin, H. W. (2000). A tabu-search-based heuristic for clustering. *Pattern Recognition*, 33(5), 849-858.
- Swanson, E. B. (1976). *The dimensions of maintenance*. Paper presented at the 2nd International Conference on Software Engineering, San Francisco, California 13-15 October 1976
- Talbi, E.-G. (2009). *Metaheuristics: From design to implementation* (Vol. 1). Hoboken, N.J.: John Wiley & Sons.
- Taylor, R. N., & Van Der Hoek, A. (2007). *Software design and architecture: The once and future focus of software engineering*. Paper presented at the Future of Software Engineering Conference (FoSE 2007), Minneapolis, MN, USA 23 - 25 May 2007.
- Tichelaar, S. (2001). *Modeling object-oriented software for reverse engineering and refactoring*. University of Berne, Berne, Switzerland.
- Tzerpos, V., & Holt, R. C. (1999). *MoJo: A distance metric for software clusterings*. Paper presented at the Sixth Working Conference on Reverse Engineering (WCRE'99), Atlanta, Georgia, USA 6 - 8 October 1999
- Tzerpos, V., & Holt, R. C. (2000). *On the stability of software clustering algorithms*. Paper presented at the 8th International Workshop on Program Comprehension (IWPC 2000) Limerick, Ireland 10-11 June 2000.
- Van Vliet, H. (2000). *Software engineering: Principles and practice* (2nd ed.). Chichester, England: John Wiley.
- Voßs, S., Fink, A., & Duin, C. (2005). Looking ahead with the pilot method. *Annals of Operations Research*, 136(1), 285-302.
- Wiggerts, T. A. (1997). *Using clustering algorithms in legacy systems remodularization*. Paper presented at the Fourth Working Conference on Reverse Engineering (WCRE'97), Amsterdam, Netherlands 6-8 October 1997.
- Xiaomin, W., Murray, A., Storey, M. A., & Lintern, R. (2004). *A reverse engineering approach to support software maintenance: Version control knowledge extraction*. Paper presented at the 11th Working Conference on Reverse Engineering (WCRE'04), Delft University of Technology, Netherlands 8 -12 November 2004
- Yang, X. S. (2008). *Introduction to mathematical optimization*. Singapore, Hackensack, N.J.: World Scientific Publications.
- Zhao, J. (1998). *Applying slicing technique to software architectures*. Paper presented at the Fourth International Conference on Engineering of Complex Computer Systems (ICECCS'98), Monterey, CA, USA 10-14 August 1998.
- Zuse, H. (1991). *Software complexity: Measures and methods*. Hawthorne, NJ, USA: Walter de Gruyter & Co.

Appendix A: CD – Enclosure

The enclosed CD embodies the following content:

The Eclipse framework including the SBRE plug-in

The folder 'eclipse' contains an *Eclipse Ganymede V. 3.4.2* that includes the SBRE plug-in. The *Eclipse* version can be copied on a computer that operates with the *Windows Vista* or the *Windows XP* operating system. After starting the *Eclipse* IDE, by executing the *eclipse.exe* executable and selecting an *Eclipse* workspace, the SBRE plug-in can be started by pressing the 'Search Based Reverse Engineering' label in the top menu.

The Evaluation Workspace

The folder 'sbre_workspace' contains the workspace, which was used for the evaluation of the SBRE component. It contains the source code of the SBRE component.

Evaluation Results

The folder 'evaluation_data' contains the results of the experiments, which have been conducted during the present research.