# *A Multi-Objective Architecture Reconstruction Approach*

Frederik Schmidt

Student-ID: 0787975

A thesis submitted to Auckland University of Technology in partial fulfilment of the requirements for the degree of Doctorate of Philosophy (PhD)

2014

School of Computing and Mathematical Sciences

Primary Supervisor: Andy Connor

Secondary Supervisor: Stephen MacDonell

# *Table of Contents*

# List of Figures

# List of Tables

# List of Abbreviations

| *Abbreviation* | *Full Description* |
|---|---|
| AEI | Additive Epsilon Indicator |
| CA | Afferent Coupling |
| CCD | Cumulative Component Dependency |
| CE | Efferent Coupling |
| DCL | Dependency Constraint Languages |
| DSL | Domain Specific Languages |
| DSRM | Design Science Research Methodology |
| EA | Evolutionary Algorithm |
| EVM | EValuation Metric |
| GA | Genetic Algorithm |
| GD | Generational Distance |
| GUI | Graphical User Interface |
| IGD | Inverted Generational Distance |
| MDG | Module Dependency Graph |
| MOEA | Multiple-Objective Evolutionary Algorithm |
| MOPSO | Multi Objective Particle Swarm Optimisation |
| MQ | Module Quality |
| NCCD | Normalised Cumulative Component Dependency |
| NDPF | Non-Dominated Pareto-Front |
| PM | Polynomial Mutation |
| RC | Relational Cohesion |
| RIA | Rich Internet Application |
| SBSE | Search Based Software Engineering |
| SBX | Simulated Binary Crossover |
| SCC | Strongly Connected Components |
| SoA | Service oriented Architecture |

# 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. Stephen MacDonell for their continued encouragement and invaluable input during this work.

I would also like to thank the staff and lecturers from the School of Computing and Mathematical Sciences, who supported me during the past years of my studies. Especially, I thank Greg Knowles and Alan Litchfield for giving me access to the SCCRL Private Cloud Infrastructure. I cannot emphasize enough that the access to this resource has been crucial for the conduction of this research. I am also extremely thankful for the financial support I received in the form of the TEC BuildIT Doctoral Scholarship and the AUT Doctoral Fee scholarship.

Furthermore, I thank my current and former postgraduate colleagues at SERL, Jacqui Finlay, Minjuan Tong, Nadia Kasto, Amjed Tahir, Bilal Raza, Da Zhang, John Graves, Michael Bosu and Sherlock Licorish for their personal encouragement and friendship during this study. In particular, I thank Jim Buchan for his support and our lively discussions.

Special thanks go out to my family, Eibe, Marianne and Talkea for their doubtless believe in my capability to finish this work. Finally, I owe my deepest gratitude to my partner, Frauke, for her patient and enormous support during this journey. You are awesome.

# Abstract

Design erosion is a persistent problem within the software engineering discipline. Software designs tend to deteriorate over time regardless of the ambitions of the development stakeholders involved. The preservation and restoration of the software design and architecture are often hampered by rapid increases in size and complexity, changing requirements and insufficient understanding of the aspired architectural design. Technical debt accumulates due to neglected refactoring and maintenance activities. A comprehensive redesign and redevelopment is often inevitable if the deterioration is not confined at an early stage.

A variety of architecture management, reverse engineering and refactoring approaches are available which can prevent or overcome this cycle of architecture deterioration. Architecture management approaches help to monitor the architecture and identify emerging violations. Reverse engineering approaches help to analyse and understand the current structure of the system, and refactoring techniques can be used to restructure the system. However, the subsequent implementation of architecture management approaches into a legacy system is a complex and time consuming endeavour which requires thorough reverse engineering, manual analysis and refactoring to re-establish a well-structured and violation-free architectural design. Hence, such approaches are typically only able to deliver partial solutions to the problems of architecture erosion, confinement of deterioration or to the challenge of architecture reconstruction.

The objective of the research presented in this thesis is to evaluate whether a Search Based Software Engineering (SBSE) approach can offer valuable support and solutions within these problem domains. The present research provides a framework to recover high-level architecture designs of software systems by structuring low-level artefacts into high-level architecture artefact configurations. The framework is implemented within a toolset to demonstrate its feasibility and to enable a thorough evaluation of the framework. The prototype features the flexible combination and configuration of SBSE

techniques with established architecture metrics and design goals to discover feasible high-level architectural designs. A variety of analysis and visualisation techniques are implemented to effectively evaluate the quality of the identified solutions. The output of this process is an architecture design classification that can be integrated seamlessly into the development process to identify emerging design deteriorations.

An important by-product of the evaluation of this research is a multi-objective evaluation artefact that enables the statistical analysis of multi-objective solution sets based on the computation of optimal Pareto-Front performance metrics. The evaluation framework enables the statistical analysis of performance snapshots by supporting the agglomeration and slicing of solution sets based on user configurations.

It has been found in this research that the application of multi-objective optimisation techniques is a feasible approach to discover high-level software architecture configurations that feature software quality attributes that would be acceptable in practise. The inclusion of conceptual target architecture models in combination with software architecture conformance metrics enables the identification of modular software architecture configurations that align with desired high-level architecture designs. The performance of different *Multi-Objective Evolutionary Algorithm* (*MOEA*) implementations and *MOEA* tunings across different architecture reconstruction scenarios and software systems has been evaluated in this thesis. It has been found that the application of *MOEA* concepts such as g*enetic algorithms, scatter search, decomposition based search*, *differential evolution* and *particle swarm optimisation* are feasible in the targeted application domain. The most prominent finding is that relaxed forms of *Pareto-Dominance* in combination with *particle swarm optimisation* search are powerful in finding promising architecture configurations in settings that feature many objectives.

# 1 Introduction

## 1.1 *Rationale*

Almost fifty years ago, the term ‘software crisis’ was first mentioned at the NATO Software Engineering Conference in 1968 (Randell, 1996). The term reflects concerns over the resistance of software systems to change and the difficulty to maintain stable software systems. The result is a predominance of inflexible and unstable software systems (De Silva & Balasubramaniam, 2012).

The paradigms, processes, tools, computational platforms and techniques in the field of software engineering have changed immensely over the past 50 years, but the problems which have been summarized under the term ‘software crisis’ still exist. Contemporary software systems that comprise any reasonable amount of functionality are invariably accompanied by a non-trivial degree of complexity (Martin, 2011). 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 static; the structure of the system changes through maintenance, requirements changes, added features and refactorings (Bosch, 2010). This creates difficulties for individuals attempting to understand the design, structures, and dependencies that comprise the architecture of a software system. As a result, realizing new requirements and maintaining a large software system is challenging.

Adding new functionality to an existing software system without considering the conceptual architecture or maintaining the integrity of the software system can result in system erosion. As a consequence software quality decreases and the system will be less flexible, less robust and harder to maintain and understand. Therefore the software maintenance cost increases. To confine or even reverse system erosion, methods of intensive reverse engineering, manual analysis and refactoring are generally required to re-establish a structured, violation-free and current architectural design. However,

development stakeholders often hesitate to engage in such complex and labour intensive tasks due to other pressing commitments and deadlines (De Silva & Balasubramaniam, 2012).

An established software engineering concept, and one method to confine erosion, is the maintenance of a high-level conceptual architecture design. Such conceptual architecture designs operate as a blue-print of the aspired design of the system. Software engineering tools exist that support the automatic compliance checking of conceptual and physical architecture artefacts. However, such architecture monitoring activities are often not implemented in the software development process and so conceptual designs are either not-defined or become outdated (De Silva & Balasubramaniam, 2012).

The challenges of the introduction of an architecture monitoring process to confine software erosion are, firstly, to find a feasible start classification of physical architecture artefacts in the conceptual architecture model and secondly to follow the implementation of refactorings to resolve architecture violations. The intention of these activities is to create a modular software system in which the physical architecture aligns with the desired conceptual architecture design. If the high-level architecture is violation-free the individual subsystems can be refactored or replaced to migrate the subsystems in line with new distribution requirements or simply to increase the design quality of the subsystems.

To date no approaches exist that support development stakeholders in the process to fit the physical architecture of a software system into the conceptual high-level architecture of a system. The reconstruction of a design based on the analysis of an existing system is complex. Multiple objectives need to be taken into consideration to find acceptable solutions. These objectives depend, for example, on factors such as the level of erosion, future plans on the refactoring or migration of the software system, agreed software quality standards and system knowledge of development stakeholders.

Little is currently known regarding the applicability of multi-objective optimisation approaches to regain software system structures. A flexible approach that offers stakeholders access to a variety of objective configurations, reconstruction strategies, optimisation algorithms and tunings can only be useful to extend the body of knowledge in this problem domain of architecture reconstruction.

## 1.2 *Objective and Contribution*

The main objective of this research is to evaluate the feasibility of multi-objective optimisation strategies when applied in the area of architecture reconstruction to identify feasible architecture classifications that can operate as a starting point for the modularisation of software systems and the containment of software erosion. Achievement of this objective is evaluated through the utilisation of a novel multi-objective architecture reconstruction framework that has been implemented in the course of this research. The framework supports recovery of high-level architecture designs of software systems by structuring low-level artefacts into high-level architecture artefacts. The output of this novel architecture reconstruction framework is an architecture design classification that can be integrated seamlessly into the development process to identify emerging design deteriorations and therefore confine the modularisation of the software system. The developed framework is implemented within a prototype to demonstrate its feasibility and utility.

The present research is informed by previous work which used *Search Based Software Engineering (SBSE)* to cluster physical source code elements by rewarding high cohesion in modules and penalising high coupling between modules (Mitchell, 2002). It is hence anticipated that the application of optimisation techniques is a feasible approach to classify physical source code elements into conceptual architecture designs and to reconstruct conceptual high-level architecture designs.

The present research suggests that the exclusive application of the rewarding of high cohesion in modules and penalising of high coupling as implemented in related research

is insufficient to reconstruct high-level architecture designs. Thus, a variety of software metrics proposed in the architecture design literature are applied to assess the quality of the design of high-level software architectures. The use of these architecture design metrics in architecture reconstruction problem scenarios is evaluated in the present research. This evaluation extends the understanding of the feasibility to employ such architecture design metrics in architecture reconstruction problem scenarios.

The reconstruction of software architectures depends on multiple factors and different requirements on the architecture configuration of a software system (e.g. distribution characteristics, module quality). Hence, the requirements on any given reconstructed architecture configuration solution are likely to differ depending on the desired modularisation, migration and refactoring objectives. It is thus anticipated that an approach that relies on a fixed configuration of reconstruction objectives would not address the challenges in the architecture reconstruction problem domain. In the present research a framework is implemented that enables the flexible definition of objective configurations to fit the objective configuration to individual architecture reconstruction requirements. The objective configuration is employed in a multi-objective optimisation approach. Hence, the developed framework combines the application of *Multi-Objective Evolutionary Algorithm (MOEA)* techniques with established architecture design metrics to discover high-level architectural designs. The developed prototype enables the employment of a variety of different *MOEAs* and *MOEA* tunings. To date there is no framework that has been shown to combine these techniques and metrics and it is contended that the research reported here extends current understandings of the possibilities and constraints in this area.

The application of multi-objective optimisation approaches results in *optimal Pareto-Fronts* (see: section 2.4.2) that feature multiple solutions. However, development stakeholders are most likely interested in the selection of one final solution that can be employed in the architecture management process. The review and selection of promising solutions based on the visualisation of multiple dimensional solution sets,

especially with more than three dimensions, is complex. Additionally, it is argued that the solution characteristics encountered in the problem domain of architecture reconstruction might not strictly underlie the assumptions of *Pareto* optimality. For example, quality assessment based on the employment of architecture design metrics (e.g. cohesion, coupling, number of cycles, number of architecture violations, distance from main sequence, cumulative component dependency) utilises simplified surrogates to assess the quality of a solution but this does not mandatorily translate to better architecture solution performance from a stakeholder point of view. To overcome such a restriction the implemented framework is able to consider the complete set of visited solutions. Hence the objective configuration rather represents a point of gravity towards which the search converges. However, it is anticipated in the present research that the stakeholder must be in charge to review and identify promising solutions. The present research hence offers a novel approach to the optimisation domain that enables stakeholders to visualise and filter multi-dimensional solution sets.

The evaluation of the performance of multi-objective optimisation approaches is in itself a complex activity. Related research in the multi-objective *Search Based Software Engineering (SBSE)* domain focuses mostly on the evaluation of the general feasibility of a developed approach and so the performance of different *MOEA* implementations and *MOEA* tunings is hardly compared (Sayyad & Ammar, 2013). Consequently, the research base to enable general assumptions on the performance of *MOEA* implementations or tunings in software engineering problem contexts is missing. The present research provides a multi-objective evaluation framework that enables the statistical evaluation and comparison of the performance of different *MOEA* tunings based on the calculation of *optimal Pareto-Front* performance indicators and analyses of the advancement in the objective space.

In summary, the main contribution of the present research is a novel framework, and a functioning prototype as an implementation of this framework, to recover high-level architecture designs of software systems to confine architectural deterioration. A

second key contribution of this research is a novel evaluation framework that has been developed to enable the statistical comparison of multi-objective solution sets. Finally, the application of the evaluation framework in architecture reconstruction problem contexts provides empirical evidence on the performance of multiple *Multi-Objective Evolutionary Algorithm (MOEA)* implementations and *MOEA* tunings in the problem domain of architecture reconstruction.

## 1.3 *Structure of the Thesis*

This thesis is structured into seven chapters. Chapter one, this chapter, describes the motivation for the research, outlines the research objective and asserts the main contributions of the research. The remaining chapters of the thesis are structured as follows: Chapter two examines relevant research in the areas of software erosion, software architecture, architecture reconstruction and *Search Based Software Engineering (SBSE)*. Chapter three discusses the applied research methodology and the design of the research. Chapter four illustrates the design and implementation of the framework in a software artefact called *Rearchitecturer*[1]. Chapter five describes the design and implementation of a multi-objective evaluation framework to statistically evaluate multi-objective optimisation solution sets. Chapter six demonstrates the application of the developed multi-objective evaluation framework in the target application domain of architecture reconstruction. Finally, Chapter seven provides conclusions gained from this research and highlights the contribution of this study to the related research fields. Furthermore, the limitations of this study are considered and recommendations for future research are given.

[1] https://code.google.com/p/rearchitecturer/

# 2 Related Work

The main objective of this work is to evaluate the feasibility of the application of multi-objective evolutionary algorithms in the domain of software architecture reconstruction. This objective is motivated by the pressing problem of software architecture erosion and the resulting complexity faced by developers when reengineering software module configurations from eroded software systems. Accordingly, this work relates to a variety of areas and problem domains within the software engineering discipline. These areas include software erosion, software architecture modelling and design, software architecture reconstruction and software architecture quality assessment, along with Search Based Software Engineering (*SBSE*) as an enabling technique for the approach developed in the present research. Relevant and contributing research in these areas are illustrated in the following sections.

## 2.1 *Erosion of Software Systems*

Erosion of software systems is not a new phenomenon and so it has been widely discussed to date. In the earliest days of the field McIlroy, Buxton, Naur, and Randell (1968) and Dijkstra (1972) stressed that the software engineering discipline was in a crisis due to inflexible software designs and consequently low maintainability. In the forty-plus years since this work was published little has changed in respect of design maintainability despite many advances in software development tools and techniques.

Lehman (1980) formulates a set of laws that explain the inevitable and continuous evolution of software systems. These laws express the phenomena of continuing change, steady development, continuing growth, increasing complexity, declining quality and self-regulation. Self-regulation describes the need of a system to adapt to its environment. Hence, even if a system satisfies the requirements at a point of time environment changes will eventually occur and require change. The existence of these laws has been empirically confirmed in a wide range of software system projects (see, for instance, the recent work of Yu & Mishra (2013)). Consequently, the erosion of

software systems is an inevitable side-effect that is likely to become evident in most non-trivial software systems.

Also, Jacobson, Christerson, Johnson and Övergaard (1992) note that software systems which are used are necessarily modified to adapt to changing domains, user and technology requirements. These modifications will gradually increase the complexity of the code base and consequently reduce both the understandability and maintainability of the software system. This increase in complexity is an unavoidable product of the business practices that enable software vendors to develop competitive software systems.

The increase in complexity combined with an often evident lack of documentation, as described by Parnas (2011) and Forward and Lethbridge (2002), hinders development stakeholders to understand and change design aspects of the system. As a consequence, uninformed design decisions might lead to a solution design that damages the architectural integrity of the system. Additionally, state-of-the-art development environments support automatic import features which put developers at risk to inadvertently include unnecessary and unwanted dependencies (De Silva & Balasubramaniam, 2012). Hence, the risk of a fast increase in coupling and complexity of software artefacts is omnipresent. The unhindered expansion of deterioration can lead to unsustainable designs, which leave only a complete redesign as a feasible option (De Silva & Balasubramaniam, 2012). However, even if a software system does not become completely inoperative, erosion will make the system more predisposed to defects, high maintenance costs and degrading performance, which has the potential to consequently lead to even more erosion. This cycle of erosion can degrade the value, usefulness and technical dominance of a software product (De Silva & Balasubramaniam, 2012).

A number of case studies point out that architecture erosion is a widespread problem in both commercial and open source software projects. Eick, Graves, Karr, Marron and Mockus (2001) depict the effect of erosion in a study of a 15-year old

telecommunication software system developed in C and C++ and featuring more than one hundred million lines of code. Eick et al. (2001) measure the impact of the implementation of change requests onto the erosion level of a system and identified a strong relationship between the required development effort to implement change requests and the level of erosion in the concerned components. It was also noted that an increase in the number of induced defects occurred during the implementation of change requests in components with a higher degree of erosion, and a stronger increase of erosion occurred based on higher initial levels of deterioration in the concerned artefacts.

Godfrey and Lee (2000) extract and analyse the architecture of the *Mozilla*[2] web browser (predecessor of *Firefox*) and identified a substantial number of undesirable interdependencies among the core artefacts of the system. Van Gurp and Bosch (2002) analyse the extracted data from Godfrey and Lee (2000) and conclude that these undesirable interdependencies might have an influence on delayed releases and also on the redevelopment of some of the core artefacts of the system. Similar findings are made on the relationship between erosion and maintenance activities with *Ant*[3] (Dalgarno, 2009), and the Linux kernel[4] (Van Gurp & Bosch, 2002).

Izurieta and Bieman (2007) describe decay or erosion as the deterioration of the internal structure of the design of a software system. This breakdown of the internal structure of a design is caused by changes that do not conform with the intended architectural patterns. Such changes can include violation of encapsulation, failure to follow pre-defined coding styles, and failure to meet quality criteria measured by inheritance depth, cyclomatic complexity and/or number of methods in class. Izurieta and Bieman (2007) define design pattern rot and design pattern grime as subclasses of

[2] http://www.mozilla.org
[3] http://ant.apache.org/
[4] https://www.kernel.org/

erosion. Design pattern rot is described as the breakdown of the structural integrity of a design pattern realization. Design pattern grime does not violate the structural integrity of a pattern, but is the build-up of artefacts in design-pattern classes that have no direct relevance in the realisation of the design pattern. In a subsequent study Izurieta and Bieman (2013) examine the extent to which software design patterns rot and accumulate grime by studying the aging of design patterns. The authors analyse the open source systems *JRefactory*[5], *AgroUML*[6] and *eXist*[7] in a case study and generate *UML* models from the three implementations at multiple time points to examine the decay of design patterns over time. Izurieta and Bieman (2013) find no evidence of design pattern rot in the examined systems, which means none of the introduced design patterns were deleted or broken. However, Izurieta and Bieman (2013) report modular grime build-up in all examined systems during their evolution. Dependencies between design pattern components increased without regard for pattern intent, reducing pattern modularity and decreasing testability and adaptability. Hence, the study supports the finding that the grime that builds up around design patterns is mostly due to increases in coupling.

The following section discusses an example in which the impact of cyclic dependencies on the quality of software systems is demonstrated. One of the fundamental design rules for a good physical design of software systems is a cyclic-free dependency structure between artefacts. Cyclic dependencies are a form of erosion and hence have a negative impact on a number of aspects of software quality such as understandability, reusability and testability (Oyetoyan, Cruzes, & Conradi, 2013). Accordingly, cyclic dependencies on any abstraction level are understood as an anti-

[5] http://jrefactory.sourceforge.net/
[6] http://argouml.tigris.org/
[7] http://exist.sourceforge.net/

pattern of software design (Martin, 2008). Figure 1 depicts a cyclic dependency between three artefacts of a software system.

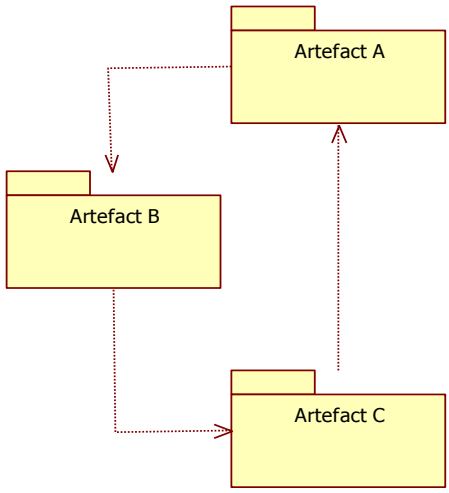


**Figure 1: Cyclic dependency design**

A further drawback of a cyclic dependency design is that the involved artefacts cannot be understood, tested or reused independently from one another. As a consequence the size and complexity of the cyclic component is unnecessarily inflated, as is the size and complexity of components using the cyclic artefact. An impact analysis to determine which artefacts will be affected by a change is significantly hindered within cyclic artefacts and a change within one artefact automatically involves all artefacts of the cycle. No specific responsibility can be assigned to the artefacts involved in the cyclic dependency. Additionally, no hierarchical order can be derived if cyclic dependencies exist between modules. Furthermore, the artefacts involved in a cycle cannot be tested independently (Beck, 2003): a test-case that calls an artefact of the cyclic artefact depends on each of the artefacts of the cyclic dependency. Consequently, the application of a hierarchal unit testing strategy, in which entities are tested following their dependency hierarchy to determine which artefact is responsible for a failing unit test, is infeasible.

In contrast, a non-cyclic dependency design does not manifest the drawbacks just discussed. Figure 2 depicts a non-cyclic dependency design.

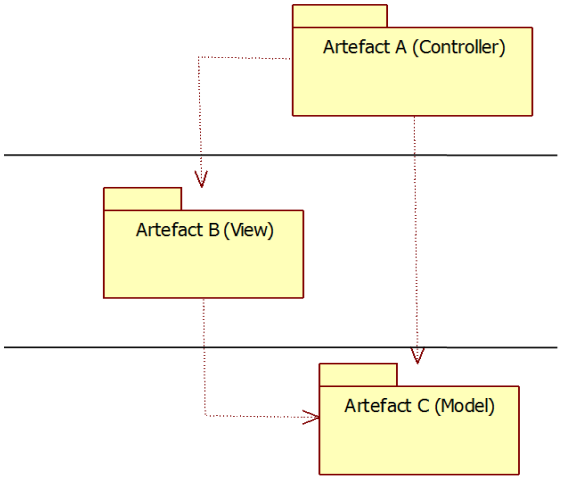


**Figure 2: Non-cyclic dependency design**

A non-cyclic dependency design enables an impact analysis to state a reliable conclusion about the artefacts that are involved in a source code change. Additionally, the artefacts of a non-cyclic dependency design can be levelled and responsibility can be assigned to the individual artefacts as illustrated in Figure 2. Finally, the thorough application of a hierarchical unit testing strategy is feasible. For example, test suites might exist to test *Artefact C* and *Artefact B*. Certainly, the test suite designed to test *Artefact B* also executes code of *Artefact C* due to the dependency from *Artefact B* to *Artefact C*. Nevertheless, the conclusion can be made that *Artefact B* is responsible for the failing of a test in the test suite of *Artefact B* if the test suite of *Artefact C* passes.

Hence, design decisions made regarding the logical design which lead to cyclic dependencies of the physical design should be avoided. Martin (2008) highlights that software designs that are not designed to be flexible with a particular focus to accommodate change tend to erode sooner. Hence, the design of software architectures is an important concern to prevent software erosion. The next section provides a brief overview on prevalent design techniques for software architectures.

## 2.2 Design of Software Architectures

The architecture of a software system is an abstract model of that system, where fine-grained entities are classified into increasingly abstract modules. An architectural view of a system therefore raises the level of abstraction, hiding details of implementation, algorithms and data representations (Bass, Clements, & Kazman, 2003). These architectural views enable different aspects of a system to be represented. For example, architectures can visualise the system from a service, application, implementation, data or process perspective (Koschke, 2008). Having a current representation of the system architecture is crucial in order to maintain, understand and evaluate a large software application (Sora, Glodean, & Gligor, 2010).

The present research focuses exclusively on the implementation perspective of software architectures. The implementation perspective is a representation of the software system on the basis of compilation units, compilation unit members, the dependency relationships between these artefacts and the aggregation of these artefacts into more coarse artefacts. The representation and analysis of software architecture on the implementation and dependency level supports the understanding of the current design of a system and is crucial in enabling the identification of design flaws (Koschke, 2008). Within the reverse engineering discipline the physical software artefacts and their dependencies are often referred to as a software architecture. However, De Silva and Balasubramaniam (2012) state that a good understanding of the physical design of the system and of the flaws in the system is by itself most likely insufficient to prevent the erosion of design if no understandings of the targeted design exist.

### *2.2.1 Physical and Conceptual Design of Software Architectures*

Jacobson, Christerson, Johnson and Övergaard (1992) suggest that a software architecture is an abstraction of domain- and enterprise-specific entities. Hence, an ideal design of a software architecture facilitates not only an aggregation of the physical implementation units into abstract subsystems but also describes the purpose of the

software system itself. Lakos (1996) also differentiates between the design of a software system into an implementation design and a conceptual design.

The implementation design describes the composition of the system on compilation units and function level. For example, an implementation design decision might be to decide if a relationship is implemented by using an inheritance or delegation pattern. From an implementation design perspective all entities exist at the same level in a shared namespace without any boundaries (Lakos, 1996).

Fowler (2002) describes the conceptual design as a view onto the system from a coarse grained and abstract level. The implementation design artefacts are partitioned into coarse grained artefacts such as packages, directories, libraries, subsystems, layers and maybe even layer groups. These coarse grained conceptual artefacts serve as containers in which to accumulate more detailed design artefacts or even finer grained conceptual artefacts from lower conceptual design levels that feature a mutual architectural design attribute (Bass et al., 2003). For example, packages accumulate compilation units, subsystems accumulate packages and layers might accumulate subsystems. From a conceptual architecture design perspective it is aspired that conceptual artefacts accumulate artefacts that comply with a certain technical, domain or environment aspect (Taylor, Medvidovic, & Dashofy, 2009). For example high-level artefacts might accumulate view, client or database functionality. Correspondingly, high-level artefacts should disclose details about the technical implementation or frameworks applied within the application (Martin, 2008). Furthermore, relationships between these coarse grained artefacts are defined as part of the conceptual design process. Relationships between implementation artefacts such as *inherits from*, *has a*, *uses relationship* are modelled as *depends on* relationships within the conceptual design (Lakos, 1996).

Malveau and Mowbray (2003) suggest the classification of the design of a software system into *Micro-Design* and *Macro-Design.* The *Micro-Design* level describes the finely

grained issues of the design and composition of low-level artefacts such as files or classes. At *Micro-Design* level, the key concerns are the provision of functionality and the optimization of performance. The *Macro-Design* level describes issues at a higher abstraction level such as system-level architecture, enterprise architecture and global systems. At the *Macro-Design* level, the main concerns lean toward management of complexity and change. Malveau and Mowbray (2003) imply that these design forces are also present at finer grains, but are not of the same importance as they are at the *Macro-Design* levels. The system-level architecture entails the grouping of artefacts on the *Micro-Design* level into abstract conceptual artefacts, but also the definition of interfaces and dependencies between these conceptual system-architecture elements.

Design patterns are an integral and accepted design instrument of state-of-the-art systems on the *Micro Design* level (Martin, 2011). Object oriented design patterns offer an agreed structural composition of low level artefacts for development problems (Gamma, Helm, Johnson, & Vlissides, 1995). Freeman (2004) highlights design patterns as instruments to facilitate experience reuse instead of code reuse. Design patterns offer a uniform solution that features high recognition value and good quality characteristics. Such good quality characteristics are, for example, easier maintainability and flexibility of designs, and reduced numbers of defects and faults (Aversano, Canfora, Cerulo, Del Grosso, & Di Penta, 2007; Di Penta, Cerulo, Guéhéneuc, & Antoniol, 2008; Porras & Guéhéneuc, 2010).

High-level design patterns exist that define the design on the conceptual architecture level. The conceptual architecture of an application can be modelled to facilitate different architecture patterns such as client/server, component based, vertical and/or horizontal layered, message bus, filter, dispatcher, *Service oriented Architecture* (*SoA*) or *Rich Internet Application* (*RIA*) (Taylor et al., 2009). These architecture styles support different domain, quality and environment requirements. Depending on the requirements and purpose of the system, multiple styles are combined to define a complete conceptual architecture model. For example, layered

architectures can be used with component-based, object-oriented or *SoA* styles. The requirements of the system determine the selection and mixture of architecture patterns. For example, a strongly distributed architecture might lead to strict layered architecture while a performance-critical system might be implemented as a transient layered architecture.

Most of the architecture styles or combinations of architecture styles can be modelled with a layered or subsystem based architecture model and allowed dependencies between these high-level artefacts. Hence, the implemented prototype developed and evaluated in the present research focuses on the reconstruction of subsystem and layered high-level architectures. Figure 3 illustrates an example of a conceptual architecture description that facilitates strict horizontal and transient vertical layering.

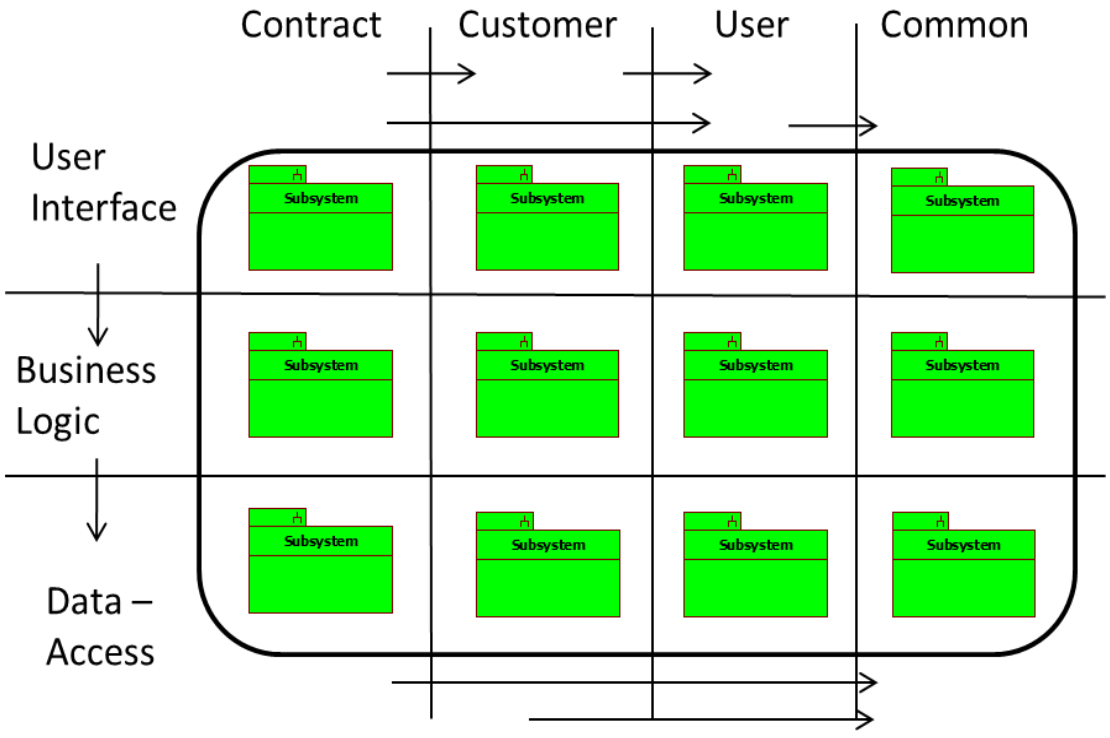


**Figure 3: Conceptual architecture model**

In the example presented in Figure 3 the horizontal layers represent the enterprise entities from a technical perspective while the vertical layers describe domain-specific applications. Allowed dependencies between layers are defined and shown as arrows between the layers. Subsystems exist within the intersection of horizontal and vertical layers and these subsystems accumulate implementation artefacts such as folders, packages and compilation units that feature functionality corresponding to their horizontal or vertical classification.

The conceptual architecture model facilitates a development blueprint for the development stakeholders. Taylor et al. (2009) highlight that the conceptual design is not only an abstraction of the logical design. It is in fact a description of the aspired design and ideally presents the purpose of the system. Martin (2008) highlights that a conceptual software architecture is a description of the application and not the technical implementation details. Fowler (2002) suggests that an ideal conceptual architecture models the domain and technical environment of the software system and delivers a framework to maintain desired quality aspects.

In spite of its asserted importance, Jansen, Avgeriou, and van der Ven (2009) and Van Heesch, Avgeriou, and Hilliard (2012) argue that the documentation of the architecture of software systems is often outdated or at least partially lost. Hence, the information about the desired design and the classification of low-level artefacts into a conceptual architecture model might be incomplete. Hence, the present research applies reverse engineering approaches that rely on the analyses of source code, as a current form of documentation, to partition software systems.

It is acknowledged that reverse engineering approaches as applied within this research have limited power to discover compositions that express the original purpose of the system. Nevertheless, the present research follows the hypothesis that the inclusion of conceptual architecture configurations in the reconstruction process is a viable instrument to find more targeted modularisations and establish a desired

modularisation in the code base. Whilst such modularisations may not provide an absolute best architecture, they offer the potential to slow or reverse the gradual degradation of the system. Based on this established modularisation, manual refactorings can be conducted to further improve the modularisation or the replacement of modules, which as a result is more straightforward due to a more organised dependency structure. Correspondingly, the next section briefly describes methods of conformance checking of physical and conceptual architecture designs.

### *2.2.2 Conformance of Physical and Conceptual Architecture Design*

Murphy, Notkin and Sullivan (2002) emphasise that the compliance of the physical architecture and the conceptual model needs to be continuously checked and that the two should be aligned as needed, to obtain a violation-free architecture. Ideally, the physical dependencies should align with the conceptual architecture model of the system (Taylor et al., 2009), although in practice this may not always be achieved. An architecture violation is therefore understood as a dependency within the physical dependency structure which conflicts with the defined dependencies of the conceptual architecture (Fowler, 2002). Identified architecture violations need to be eliminated to obtain a modular physical code base and also to obtain the desired architectural design within the physical code base (Fowler, 2002). Figure 4 demonstrates an example of a conceptual architecture model that exhibits a physical dependency that conflicts with the defined conceptual architecture model.

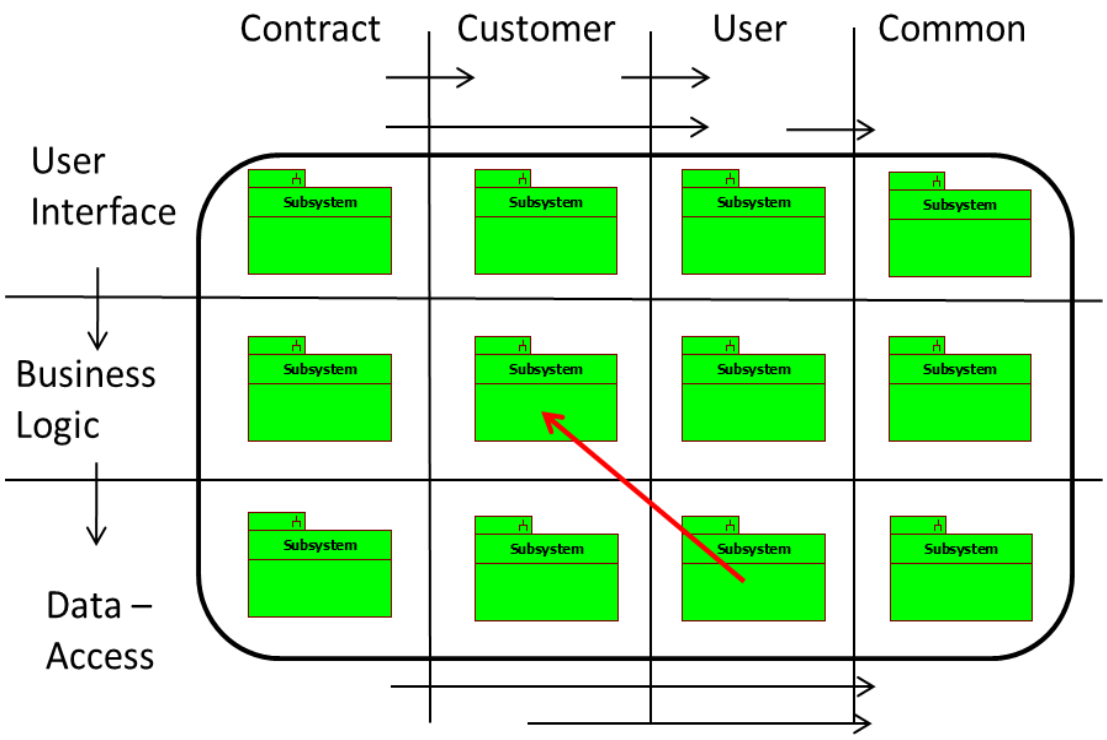


**Figure 4: Conceptual architecture model with architecture violation**

The depicted dependency violates the horizontal conceptual layering rules with an access from *Data-Access* to *Business-Logic* and the vertical conceptual layering with an access from *User* to *Customer*. Physical dependencies that do not conflict with the conceptual architecture model are not shown in Figure 4.

Architecture violations occur if the conceptual and physical architectures drift apart. Additionally, violating dependencies create cyclic dependencies on layer, subsystem and maybe even package level. Correspondingly, the consideration of the conceptual architecture model gives access to a variety of software architecture metrics that are relevant to assess the maintainability and modularisation of a software system. Section 2.3 describes the metrics that are said to be particularly relevant for the conduct of this research. Some of these metrics assess the classification of the physical source code artefacts in the conceptual architecture.

Depending on the nature of the architecture violations, different refactoring techniques can be applied to resolve them (Fowler, 2002). For example, if a compilation unit has been created in the wrong package, a simple *move compilation unit* refactoring might be sufficient to resolve the existing architecture violation. However, if a compilation unit features two responsibilities which belong in two different artefacts of the conceptual architecture model, the code segments that entail these responsibilities need to be decomposed and put into new and/or existing artefacts within suitable high-level artefacts of the conceptual architecture. Low-level refactorings such as *extract method*, *move method* and *move field*, which are capable of changing the composition and consequently the dependency structure of the concerned compilation units themselves, need to be applied to eliminate such architecture violations (Fowler, 1999). On the other hand, changes in technical implementation or domain aspects of the system might require a review of the conceptual architecture model (Martin, 2011).

In order to resolve architectural violations using refactorings there is the prerequisite to identify if any architecture violations exist. Different approaches exist to help in the process of identifying architecture violations. Reflexion Models (RM) and Domain Specific Languages (DSL) are the most popular methods to define conceptual architectures and detect architectural violations with the aim to confine the erosion of software architectures (Maffort, Valente, Anquetil, Hora, & Bigonha, 2013).

Murphy and Notkin (1997) and Murphy, Notkin, and Sullivan (2002) depict the *Reflexion Model* as a framework to prevent the deterioration of software architectures. The reflexion model features a conceptual architecture as a set of subsystems. Allowed and forbidden dependencies between the conceptual artefacts are defined and physical implementation units are mapped into the subsystems. Reflexion models can detect the absence of dependencies, which are mandatory dependencies in the conceptual model that are not present in the physical dependency model, and divergences, which are forbidden dependencies that are not allowed based on the conceptual model but that

are present in the physical model. Figure 5 illustrates the general process of the application of the *Reflexion Model*.

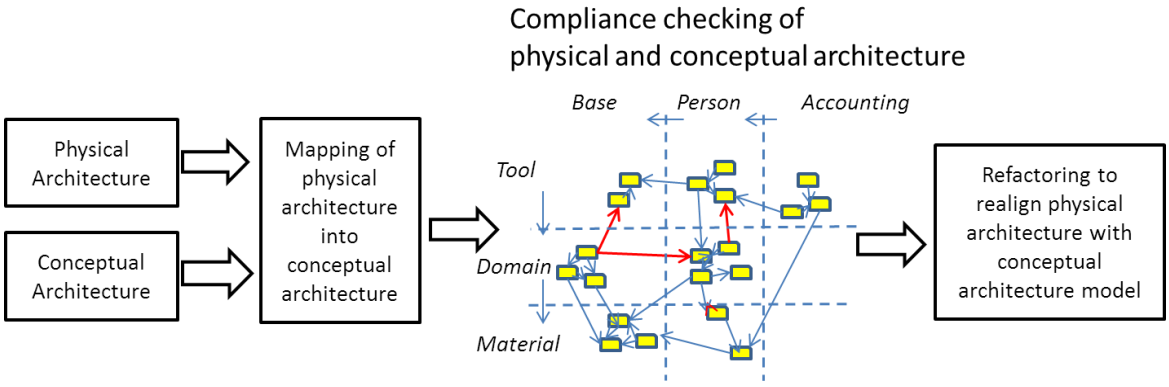


**Figure 5: Reflexion model process (adapted from Murphy et al. (2002))**

The application of *Domain Specific Languages* (*DSL*) is an extended approach to facilitate the requirement of defining a conceptual architecture and to execute compliance monitoring (Passos, Terra, Valente, Diniz, & Mendonça, 2010). *DSLs* that focus on architecture conformance provide means for software architects to express in a customized syntax the constraints defined by the planned architecture. However, the mapping from physical implementation artefacts follows a similar procedure as the mapping within the *Reflexion Model*. A specific implementation of a *DSL* in the domain of architecture conformance description is the *Dependency Constraint Language* (*DCL*) (Passos et al., 2010; Terra & Valente, 2009; Terra, Valente, Bigonha, & Czarnecki, 2012). Terra and Valente (2009) present an alphabet and construction rules of the *DCL* as illustrated in Figure 6.

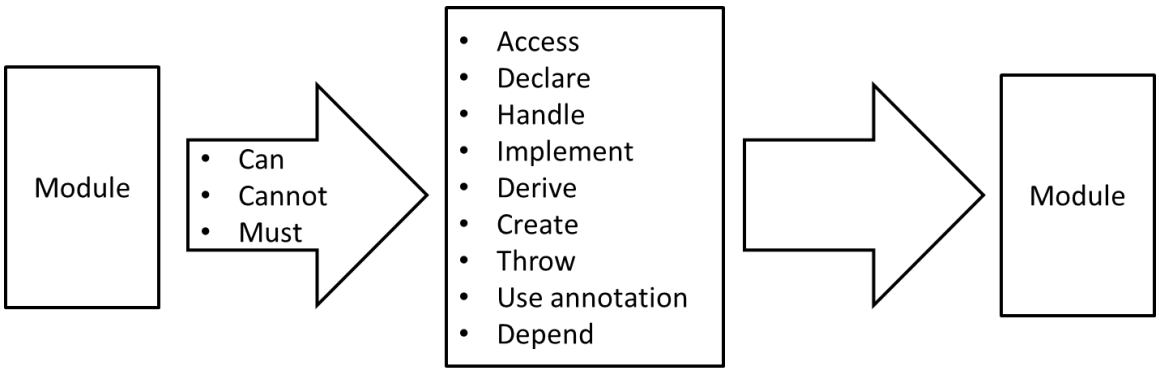


**Figure 6: Alphabet and construction rules of DCL (Terra and Valente, 2009)**

A variety of commercial and open source systems exist (Ducasse & Pollet, 2009; Duszynski, Knodel, & Lindvall, 2009; Sangal, Jordan, Sinha, & Jackson, 2005) to support the tasks of architecture monitoring and conformance checking. *Dependometer*[8], *Lattix*[9], *Sotograph*[10] and *XRadar*[11] are examples of such architecture management systems. These architecture monitoring tools feature mixed implementations of the ideas of the *Reflexion Model* paired with individual techniques of *DSL*. The objective of each of these systems is to prevent and contain the erosion of software systems. Each of these tools requires the definition of a conceptual architecture and a manual mapping of the physical architecture into the conceptual or desired architecture. Within the mentioned architecture management systems the mapping is implemented by assigning packages or package patterns to subsystem, layer and/or vertical slice artefacts.

A drawback of the application of the *Reflexion Model* and *DSL* is that both approaches require the definition of the conceptual architecture model and mapping of the physical architecture into the conceptual architecture model to reveal the whole spectrum of absences and divergences that can exist in the source code. Hence, the task to (perhaps retrospectively) define the conceptual architecture and define the mapping

[8] http://source.valtech.com/display/dpm/Dependometer
[9] http://*www.lattix.com/*
[10] http://*www.hello2morrow.com/products/sotograph*
[11] http://*xradar.sourceforge.net/*

of the physical artefacts into the conceptual architecture model requires a sound understanding of the aspired system design and the application domain of the system. These tasks can be challenging for development stakeholder, particularly if the system is long-lived and/or is not well documented.

Maffort et al. (2013) and Taylor et al. (2009) highlight that, despite the availability of approaches to conduct compliance checking of physical architecture and conceptual architecture, only a limited number of software projects thoroughly apply such techniques. Consequently, without a rigorous compliance checking and refactoring of the physical architecture, the conceptual and physical architectures tend to drift apart for many software systems.

## 2.3 *Quality Assessment of Software Architectures*

It has long been held that the assessment of the quality of a software artefact is a complex endeavour (Jones, 1997; Kan, 2002; Mordal et al., 2013). One way to evaluate the quality of software artefacts is the application of software metrics (Fenton & Neil, 2000). Software metrics are surrogates that represent a quality aspect of an artefact under consideration as a numerical value. The present research implements a search based driven software modularisation approach. The concepts suggested in Harman and Clark (2004) to utilise software metrics as a fitness function is implemented in this research to evaluate the generated architecture configurations and enable the navigation through the search space.

This section describes software design metrics that are particularly relevant for the assessment of software architecture designs. The described metrics have been implemented within the developed prototype, which is described in Chapter 4. Additionally, the presented software design metrics are utilised during the evaluation of the targeted objectives of the present research. The following sections describe the concepts that underlie these metrics and the reasoning for their application in the present research. Some of the illustrated metrics (cohesion and coupling) have already

been applied in related research activities and Section 2.4.3 highlights such research projects in more detail. Nevertheless, as far as it can be determined the combination of metrics within a multi-objective search approach has not been attempted.

### *2.3.1 Cohesion Metrics*

There is general consensus within the software engineering community that high cohesion within artefacts is a desired design goal and an indicator of good design (Martin, 2008). The reasoning that underlies this design principle is that an artefact should focus on its purpose, i.e. only entail functions and structures that relate to the intention of the artefact (Meyer, 1988). The benefit of such a design is that artefacts that feature high cohesion are easier to understand, test and maintain. On the other hand artefacts with low cohesion would entail a variety of responsibilities and are consequently harder to understand, test and maintain.

Various metrics exist to measure cohesion within classes. Cohesion metrics that operate on higher abstraction levels such as package, subsystem and system level are of particular relevance to address the objective of the present research. Gui and Scott (2006) suggested to measure cohesion on higher abstraction levels to evaluate component re-usability, and as such they defined the cohesion of a high-level artefact as the mean of the cohesion measures of all members of the artefact. Martin (2000) proposed a package level metric called *Relational Cohesion (RC)* that expresses the interconnection of the low-level artefacts that are included in a high-level artefact as a numerical value. *RC* is calculated as a ratio of the number of dependencies that are internal to the high-level artefact (i.e. dependencies that do not connect or refer to other artefacts outside the high-level artefact for which the *RC* metric is calculated) and the number of low-level artefacts that are included in the high-level artefact. A shortcoming of the *RC* metric is that the best *RC* measure for an artefact is achieved if the dependency graph within the artefact is complete. However, such a graph will most definitely feature cyclic dependencies. Cyclic dependencies on low design levels, such as class level, are often unavoidable and acceptable to a certain degree. However, utilising

the *RC* metric on a subsystem level with packages as the member artefacts is more concerning, as the *RC* metric would allow cyclic dependencies between packages which would negatively impact various quality attributes of the system partitioning such as understandability, reusability and testability.

The *RC* metric is applied as the cohesion measure of choice within the present research despite the mentioned challenges linked with its application. Hence, it is important (and feasible) to include optimisation goals that battle these shortcomings of the *RC* metric. It is understood that it would be promising to apply cohesion implementations which consider different aspects of cohesion and not only internal artefact dependencies in a reconstruction approach. These measures might be able to capture the original principles of cohesion more accurately. However, having no access to corresponding implementations and/or inadequate performance to apply these metrics in a search based context hinders the implementation of such cohesion implementations in the present research.

### *2.3.2 Coupling Metrics*

Coupling measures the strength of dependency between artefacts (Meyer, 1988). Consequently, coupling gives an indication to what degree a program artefact relies on each one of the others. Low or loose coupling indicates that the source code is organized in such a way that it features no strong dependencies between each of its members (Szyperski, Bosch, & Weck, 1999).

The dependencies of an artefact can be differentiated into afferent and efferent dependencies (Martin, 1994). Afferent coupling (*CA*) measures the number of artefacts that depend upon the artefact under consideration. *CA* is essentially an indicator of how much responsibility an artefact has. *CA* can help to reveal the effects that a change to an artefact will have on depending artefacts. Efferent coupling (*CE*) describes the number of artefacts that the artefact under consideration depends upon. *CE* is an indicator of

the artefact's independence. *CE* can be a useful indicator to evaluate the degree to which outside changes will affect an artefact.

Design techniques to achieve low coupling within software systems are, for example, prevention of cyclic dependencies, maintenance of high-level dependency structures, referencing of interfaces instead of concrete types and application of dependency injection frameworks. Previous research that applies *SBSE* in the area of software modularisation, as is described in more detail in section 2.4.3, focuses on reducing the coupling between artefacts by reducing the number of direct dependencies between artefacts. These studies suggest that the reduction of edges between artefacts is a feasible objective to modularize low-level artefacts into modules at higher abstraction levels. One example that underpins the limitations of the application of the number of direct dependencies between artefacts modules, as described by Mitchell and Mancoridis (2001a), Mahdavi, Harman, and Hierons (2003) and Praditwong, Harman, and Yao (2011), is that a solution is considered an improved solution as long as the number of edges between the existing artefacts is reduced. This contradicts design principles of software architecture that allow high-level artefacts to use other high-level artefacts and forbid or limit access to other artefacts (Lakos, 1996). Correspondingly, from an architecture design point of view, it has to be recognised that the existence of dependencies between high-level artefacts is not necessarily an anti-pattern to indicate high coupling. Therefore, the strength of coupling is not defined by the number of dependencies between artefacts but rather which artefacts, and how an artefact accesses another artefact.

The present research promotes that other aspects of coupling are also important in terms of reconstructing an accurate and useful high-level architecture model. Such aspects are reduction of cyclic dependencies between artefacts, organisation and grouping of members within high-level artefacts, consideration of hierarchical structures and access permissions of high-level artefacts. This research proposes that the consideration of such aspects supports the reconstruction of a high-level architecture

model that facilitates understandability, testability and reusability. Nevertheless, it is acknowledged that the design of the prototype developed in the current research is not able to change the implementation and dependency structure below compilation unit level. Hence, the reconstruction can only take place within the limitations of such an approach and the design of the reconstructed system.

The following sections present an overview of coupling metrics that are utilised within the evaluation of the developed architecture reconstruction prototype and which consider aspects such as dependency flow and structure instead of relying on the exclusive reduction of dependencies between artefacts. Additionally, the sections outline how these coupling metrics contribute to the implementation of the objectives of the present research.

### *2.3.3 Cycles and Architecture Violation Metrics*

As illustrated in Section 2.1, the existence of cyclic dependencies limits reusability, testability and the impact analysis of changes in the involved system artefacts. Empirical evidence supports that cyclic dependencies are evident in almost all non-trivial software systems on lower abstraction levels (Melton & Tempero, 2007). Recent research empirically underpins that most cycles on compilation-unit level do not deteriorate the testability and reusability of software systems (Al-Mutawa, Dietrich, Marsland, & McCartin, 2014). Falleri, Denier, Laval, Vismara, and Ducasse (2011) argue that the composition of individual cycles should be considered, to assess the impact on the quality of a software system. For example, longer cycles have a more negative impact on the structural quality of a software system. However, an established principle of good architectural design is that software architectures feature a cycle-free design on package and higher abstraction levels to support quality attributes such as testability, reusability and understandability (Martin, 2008). Nevertheless, the optimisation of architecture compositions towards designs that feature a small number of cycles is a worthwhile objective and hence pursued in this research project. The application of more specific cycle detection metrics as suggested in Falleri et al. (2011) is certainly

promising but an objective for future research that should be pursued once the general feasibility of cycle detection metrics in multi-objective optimisation approaches has been evaluated.

Both Seng, Bauer, Biehl, and Pache (2005) and Abdeen, Ducasse, Sahraoui, and Alloui (2009) apply cycle detection metrics within a single objective search based modularisation problem. These approaches are capable of identifying cycles between modules that feature a direct cyclic dependency between two directly connected modules. Hence, the approach presented in Seng et al. (2005) and Abdeen et al. (2009) are not able to resolve cycles that span across more than two modules. Naturally, longer cycles have a more negative impact on the quality of a software system. Hence, the development of an approach that also captures cycles with more than two elements is an important advancement of previous approaches to assess the quality of a design.

A number of early approaches for identifying the absolute number of elementary cycles within a graph have been evaluated to inform the current work (Johnson, 1975; Szwarcfiter & Lauer, 1976; Tarjan, 1973; Tiernan, 1970). However, within the development of the present research it has been found that the computational complexity on lower artefact levels (e.g. compilation unit level) is too high to justify the application of these cycle detection approaches within an optimisation approach. One possible explanation is that the worst case computational complexity of these cycle detection algorithms is evident in complete graphs. Hence, within the application of *SBSE* approaches that start with random initialised sets of solutions it is likely that the solutions feature high connectivity that then causes long evaluation run times of the applied algorithm. Hence, the search is incapable of creating a sufficient number of offspring in an acceptable amount of time to identify solution attributes that are relevant to create solutions with a lower number of elementary cycles. This suggests that the application of algorithms that calculate the absolute number of elementary cycles is infeasible in a search based driven context.

Nevertheless, Tarjan (1972) introduced an algorithm to calculate the *Strongly Connected Components (SCC)* of a graph. A *SCC* is defined as a set of vertices in which a path exists from any vertex to any other vertex of the *SCC*. Hence, all members of a *SCC* feature cyclic dependencies between each other. A vertex that is not a member of a cycle forms a strongly connected component by itself. From a software design perspective it is desired to have only *SCCs* that include one single member.

The worst case complexity to calculate the *SCC* of a graph is O(|V|+|E|) for a graph G(V,E) with V vertices and E edges. This low complexity makes the number of *SCCs* a promising objective metric for application in the present research. Correspondingly, an objective metric has been implemented as part of this research. This objective enables to employ an optimisation based on the number of *SCCs* that feature more than two members on layer, subsystem and package levels. More detail on the employment of this and other objectives in the *Rearchitecturer* prototype can be found in Chapter 4.

Another aspect to support the modularisation of software systems is the reduction of architecture violations. Layered and subsystem architecture designs can be defined within the conceptual architecture model. Additionally, allowed and forbidden dependencies between the defined layers and subsystems can be defined. These high-level layer and subsystem configurations represent a conceptual target-architecture model that can be considered during the architecture reconstruction process that is implemented in the *Rearchitecturer* prototype. The low-level artefacts are classified into the defined architecture model during the reconstruction process. Architecture violations occur if the classification of a software system does not match the targeted architecture design (compare section: 3.1). The developed approach offers the detection of architecture violations on subsystem, package, compilation-unit and type level. The number of architecture violations can be utilised as an objective to find configurations that align with the defined architecture model. It has to be raised as a limitation of the proposed approach that such an objective will attempt to resolve every cycle and architecture violation even if the origin of such violation has its origin in a

micro design implementation that does not conform with the targeted conceptual architecture model.

### *2.3.4 Structure Assessment Metrics*

The present research addresses the reconstruction of a software architecture design by classifying low level-artefacts (e.g. compilation units and/or packages) into artefacts of higher abstraction levels (e.g. packages, subsystems and layers) depending on the reconstruction strategy. Previous research in this area focused on the grouping of elements into the next highest abstraction level (Abdeen et al., 2009; Mitchell & Mancoridis, 2006; Praditwong et al., 2011). When dealing with different abstraction levels it is worth considering the structure of the elements within high-level artefacts and to discuss the impact on quality aspects of the reconstructed architecture. Lakos (1996) introduced a set of metrics that focus on more expansive aspects of the dependency structure and that correspondingly might be valuable within the evaluation of the present research. The following sections depict and discuss Cumulative Component Dependency (CCD) and *Normalized Cumulative Component Dependency* (*NCCD*) as two promising metrics to be applied within a multi-objective architecture reconstruction approach.

The *Cumulative Component Dependency (CCD)* metric introduced by Lakos (1996) features promising aspects to be applied within a search based driven architecture reconstruction approach. *CCD* is defined as the sum of all depending artefacts of all members of an artefact (Lakos, 1996). Figure 7 illustrates three examples of the calculation of the *CCD* metric. Each of the entities represents a software artefact. These artefacts can be a compilation unit, a package or even a subsystem or layer. Each artefact depends on itself and the number of dependant artefacts. The *CDD* for a dependency graph is defined as the sum of the number of dependant artefacts of each artefact of the dependency graph.

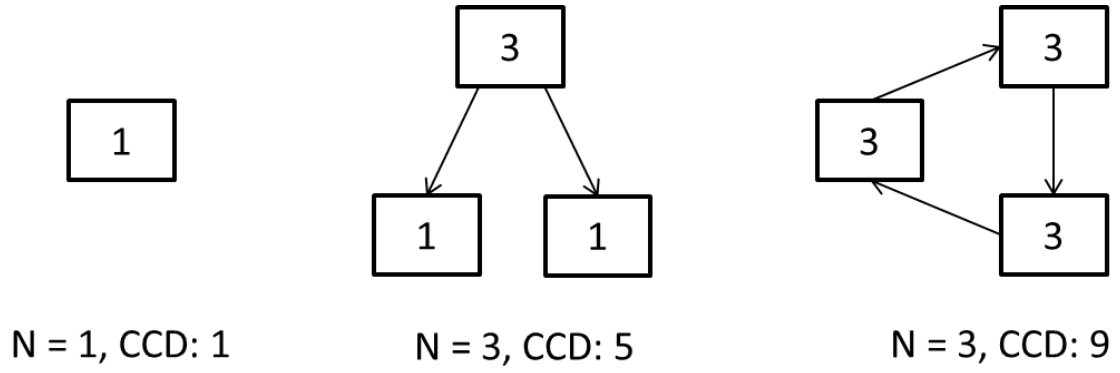


**Figure 7: CCD calculation**

In introducing the *CCD* metric Lakos (1996) originally focuses on measuring aspects that impact link and compile time. However, it can be reasoned that the *CCD* metric and its extensions have relevance to assessing aspects of software design quality. Figure 7, for example, illustrates that the *CCD* metric penalises the existence of cyclic dependencies. The dependency graph with 3 elements and a cyclic dependency leads to a *CCD* value of 9 whereas the dependency graph with no cyclic dependency produces a *CCD* value of only 5. Therefore, minimizing *CCD* for a given set of components is a design goal (Lakos, 1996).

As noted above, the approach that has been developed within the present research is unable to change the dependency structure of artefacts at compilation unit level as no transformations are supported in the present research to change the composition of compilation units. Hence, the *CCD* metric measures of individual compilation unit artefacts remain unchanged. Nevertheless, the *CCD can* be calculated for higher abstraction levels. Figure 8 pictures two example classifications of an identical dependency graph featuring the classification of seven compilation-unit artefacts into three package artefacts. The example assumes that the packages are also assigned to an artefact of higher abstraction, such as a subsystem or a layer. Figure 8 shows that different partitionings of low-level artefacts can lead to different *CCD* metric values at higher abstraction levels.

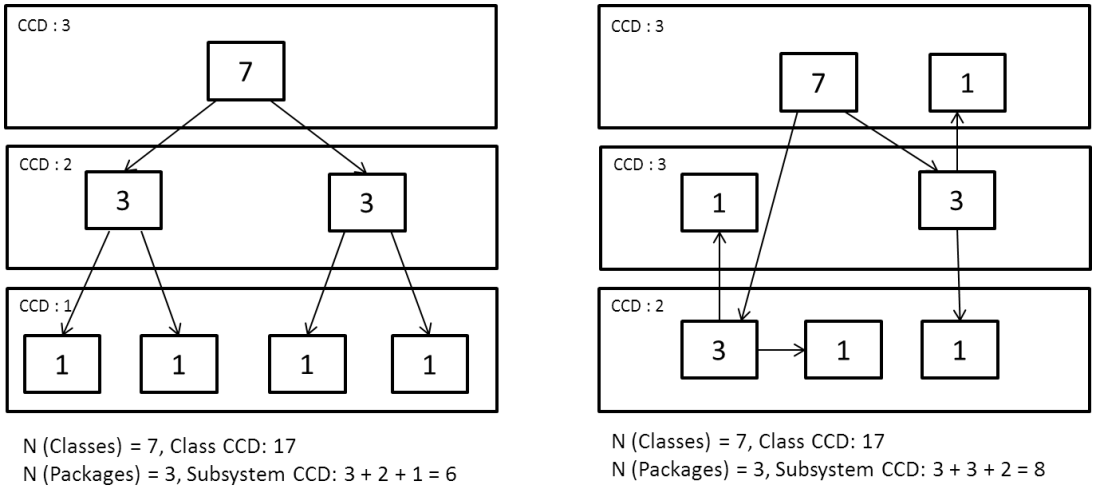


**Figure 8: Impact of cyclic dependencies on CCD**

The example illustrated in Figure 8 further suggests that generally a minimisation of the *CCD* measure rewards horizontally levelled and cyclic free designs. Hence, the *CCD* metric is a promising metric candidate for the optimisation of high-level-architecture designs. However, one drawback of applying *CCD* within a search based driven architecture reconstruction approach is that the optimal or lowest possible *CCD* is achieved for a design if all low-level artefacts are assigned into one of the high-level grouping artefacts and respectively no dependencies between modules exist. Figure 9 illustrates such an extreme classification in which all low-level artefacts are assigned into *Subsystem1*. None of the low-level artefacts are assigned into *Subsystem2* and *Subsystem3*. Such an unfeasible modularisation outcome would be likely if the *CCD* metric is utilised as a *single* objective optimisation goal.

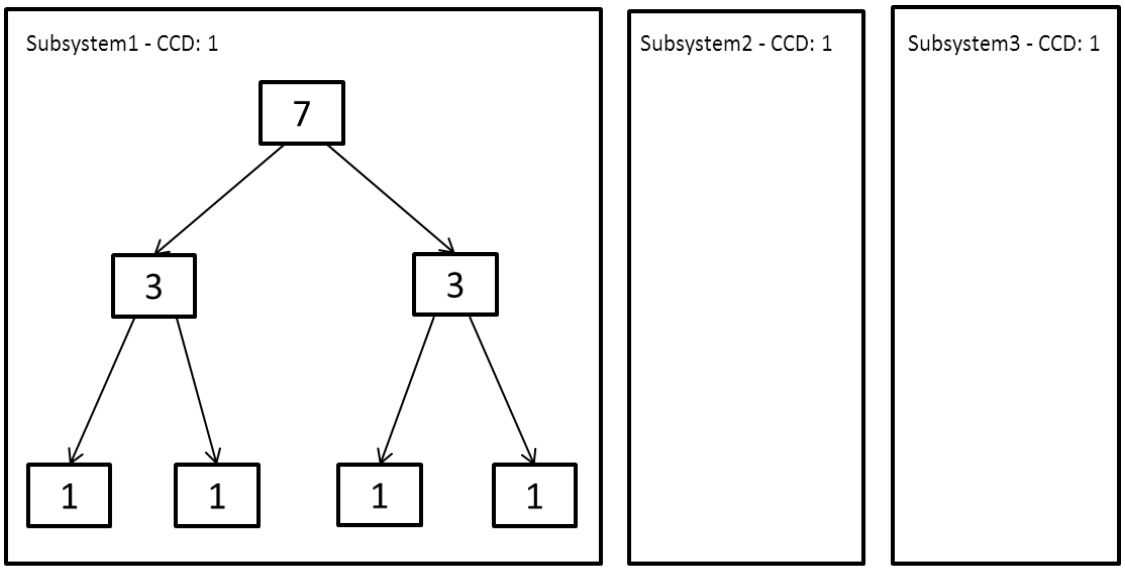


**Figure 9: Impact of unbalanced classification on CCD**

Three feasible approaches can be employed within the *Rearchitecturer* prototype to overcome the rewarding of extremely flat designs as a disadvantage of the application of the *CCD* metric as an objective setting. First, other objectives such as maximising the number of members in the high-level artefact or minimizing the standard derivation of *CCD* of all high-level artefacts can be employed to battle such undesired extreme solutions. The downside of this approach is that the optimal set of solutions (compare *optimal Pareto-Front* in section: 2.4.2) might contain solutions that feature empty high-level artefacts. Hence, the *Pareto-Front* has to be manually reviewed to exclude such unfeasible solutions. Another method to exclude extremely flat solutions is the definition of constraints that penalise solutions that feature unwanted metric attributes. Finally, the *Normalized Cumulative Component Dependency* metric also introduced by Lakos (1996) as an extension of the *CCD* metric offers partial support to overcome the shortcoming that the *CCD* metric rewards extreme solutions which feature the grouping into one single high level-artefact.

The previous section depicted that the *CCD* metric enables the evaluation of software assemblies based on the number of dependent components. Low measures of *CCD* indicate cyclic free and rather flat hierarchical designs that feature good testability, understandability and reusability. High *CCD* measures, on the other hand, indicate rather vertical designs with high coupling. That said, the *CCD* metric is not independent of system size. Bigger systems will feature higher *CCD* measures, which will consequently hinder the assessment of the hierarchical quality and shape of a system based on the *CCD* measurement alone. Lakos (1996) introduces the *Normalized Cumulative Component Dependency (NCCD)* metric to eliminate the effect of system size. The *NCCD* metric calculates the ratio of the *CCD* of a system containing *N* components and the *CCD* of a balanced binary tree-like system with the same number of components. As an example, *Subsystem1* in Figure 9 entails a balanced binary tree of low level artefacts. The formula to calculate the *CCD Balanced Binary Tree* metric for an artefact with N elements is presented in Lakos (1996) as:

$$\text{CCD Balanced Binary Tree (N)} = (N+1)\log_2(N+1) - N$$

The *NCCD* metric as a ratio of the *CCD* of an artefact with N elements and the corresponding balanced binary tree with N is calculated as follows:

$$\text{NCCD(artefact)} = \frac{CCD\ (artefact)}{\text{CCD Balanced Binary Tree (N)}}$$

Lakos (1996) states that a system that features a *NCCD* of less than 1.0 indicates a more loosely coupled design and a system with a *NCCD* of greater than 1.0 indicates a more vertical and more tightly coupled design. The circumstance that the *NCCD* metric considers the *CCD* of a balanced binary tree enables valuable conclusions to be drawn about the shape of the dependency graph within an artefact. For example, the coupling hierarchy of an artefact will be similar to the coupling hierarchy within a balanced binary tree if an assembly features a *NCCD* measure that is similar to 1.0. Only a few low-level artefacts within such a design will feature a high afferent coupling and a high

responsibility. Additionally, the lower the level of the component within the tree-structure the lower is the ratio of afferent coupling to efferent coupling. No components within the assembly feature high efferent and high afferent coupling within a *CCD* structure with a balanced binary tree structure. Consequently, a balanced binary tree structure is a useful target structure to find assemblies that support favourable quality aspects and also to prevent extreme horizontal designs. Hence, the application of the NCCD metric is promising within an optimisation approach to discover designs that feature such preferable structural attributes.

It needs to be kept in mind that the original purpose of the *NCCD* metric is to assess the quality of a software system based on the assessment of low-level dependencies. As addressed previously, the approach developed within the present research is unable to change the dependency structure below compilation unit level. Consequently, the approach is unable to change actual re-use or understandability at the source code element level. Nevertheless, the application of the *NCCD* metric is promising in terms of creating assemblies of low-level artefacts that support good reusability, understandability and testability based on the initial system design. However, it is understood that these reconfigurations can only operate within the parameters of the original source code design of the system.

Another metric to assess the structure of a software design is the *Distance* metric. The *Distance* metric describes the distance from the main sequence of the *Abstractness (A)* and *Instability (I)* metric of an artefact (Martin, 2000). *Abstractness* is the ratio of the number of abstract classes and interfaces in the artefact under consideration to the total number of classes within the artefact. *Abstractness* has a range of 0.0 to 1.0. An *Abstractness* of 0.0 indicates a completely concrete package and an *Abstractness* of 1.0 describes an artefact that only entails abstract artefacts. *Instability* describes the ratio of *Efferent Coupling (CE)* to total coupling (*CE* + *CA*) and is an indicator of the artefact's resilience to change. The range for *Instability* is 0.0 to 1.0. An *Instability* measure of 0.0 indicates that an artefact does not have any efferent couplings and consequently is

highly stable. An *Instability* measure of 1.0 indicates that a component has only efferent couplings and consequently is a highly unstable artefact.

The *Distance* metric attempts to balance the metrics of *Abstractness* and *Instability* by drawing the *Main Sequence* as a straight line on the *Cartesian* coordinates X=0 and Y=1 to X=1 and Y=0 in which *Abstractness* and *Instability* are described on the Y- and X-Axes. The Distance metric is expressed as a numerical value of the distance from this main sequence. Figure 10 illustrates the concept graphically.

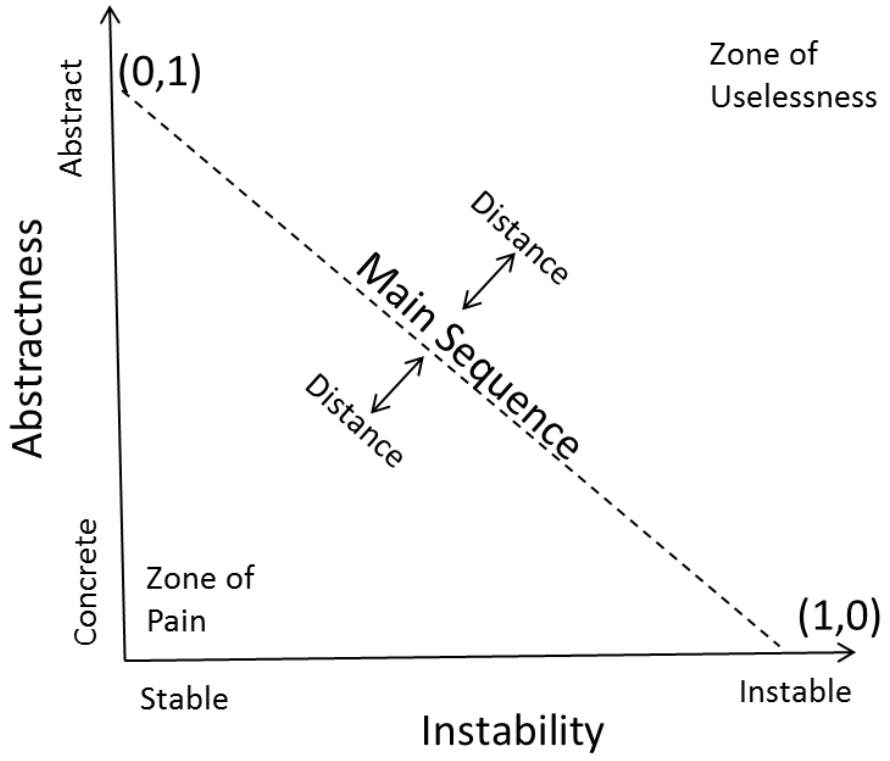


**Figure 10: Distance from main sequence (adapted from Martin (2000))**

An artefact directly on the main sequence is optimally balanced with respect to its abstractness and stability. The range for this metric is 0.0 to 1.0. A *Distance* of 0.0 indicates an artefact that is squarely on the main sequence and features a balanced mix of stability and abstractness, whereas a *Distance* of 1.0 indicates an artefact that is as far from the main sequence as possible and consequently is unbalanced in respect to

*Abstractness* and *Instability*. Martin (2000) defines the zones of extreme distance as the "*zone of pain*" (*I* and *A* both close to 0) and the "*zone of uselessness*" (*I* and *A* both close to 1.0). Assemblies that are close to the "*zone of pain*" are concrete and stable and, according to Martin (2000), potentially hard to maintain. The other extreme *"zone of uselessness"* describes assemblies that are abstract and unstable and thus are potentially useless (Martin, 2000). The extreme configurations are either completely abstract or stable (x=0, y=1) or completely concrete and unstable (x=1, y=0). However, it is an unlikely situation that all artefacts within a code base have *Abstractness* and *Instability* values of 1.0 or 0.0. Hence, most artefacts will feature values between the two. By monitoring the *Distance* from the *Main Sequence* metric one can identify artefacts that are becoming unbalanced. The *Distance* metric has potential merit within an architecture monitoring scenario in terms of identifying artefacts that are becoming unbalanced. However, a partial objective of the present research is to determine the utility of software architecture metrics when applied within an architecture reconstruction approach. The application of the *Instability* metric as an optimisation goal is certainly feasible within an automatic architecture reconstruction approach. Even if *Instability* has not been applied directly as an objective within other *SBSE* software clustering and modularisation approaches, other research has applied diverse configurations of coupling successfully. Thus, the contribution of the explicit application of the *Instability* metric is unlikely to produce deeper insights. The application of the *Distance* metric as well as the *Abstraction* metric within a single objective architecture reconstruction scenario is not promising as arbitrary classifications that do not consider the dependencies between interfaces and abstract artefacts and their concrete implementers will unlikely produce useful results. However, Amoui, Mirarab, Ansari and Lucas (2006) utilise the *Distance* metric as a fitness function within a single objective *Genetic Algorithm (GA)* implementation to include design patterns in UML representations. It is suggested in the present research that the application of multiple software metrics in a multi-objective optimisation approach is a promising approach to create solutions that exhibit good performance in multiple solution aspects. The

*Distance* metric is an example metric, whose application in a single-objective approach is probably not very useful but is a potentially useful objective candidate in a multiple objective approach in which other metrics assess quality aspects that are neglected by the *Distance* metric.

## 2.4 *Search Based Software Engineering (SBSE)*

The present research utilises Search Based Software Engineering (*SBSE*) in the domain of software architecture reconstruction. This section briefly sets out the fundamental characteristics and concepts of *SBSE*. Furthermore, aspects of *Evolutionary Algorithms (EA)* and *Multi-Objective Evolutionary Algorithms (MOEA)* are highlighted given the focus of this research. Finally, the most pertinent literature that applies *SBSE* techniques in the domain of software partitioning and modularisation is discussed.

Harman and Jones (2001) introduced *SBSE* as the application of metaheuristic algorithms to solve linear optimization problems in the domain of software engineering. They acknowledge that a vast number of application areas for metaheuristics in the domain of software engineering might exist in which the computational complexity to achieve an optimal solution would be very high if the solution space (i.e. the number of potential solutions) is large, or that there may even be no optimal solution which outperforms any other solution in all performance aspects. However, it might not necessarily be required to identify the *absolute* optimal solution for that particular problem. One of the key assumptions of *SBSE* is that within the solution space good solutions exist besides the optimal solution, which can sufficiently satisfy the requirements of stakeholders. In these problem domains metaheuristics can support the discovery of good solutions relatively fast. A good or nearly optimal solution that can be found more quickly than the optimal solution may be sufficient for stakeholders.

Since the original proposal *SBSE* has emerged as a vibrant research topic with evidence in the literature showing that the approach is widely applicable across the whole spectrum of software engineering lifecycle activities e.g. requirements

engineering (Bagnall, Rayward-Smith, & Whittley, 2001), release planning (Brasil, da Silva, de Freitas, de Souza, & Cortés, 2012; Greer & Ruhe, 2004), project planning and estimation (Burgess & Lefley, 2001; Connor & Shah, 2014; Rodríguez, Ruiz, Riquelme, & Harrison, 2011; Sarro, 2011), refactoring and maintenance (Harman & Tratt, 2007; Hemati-Moghadam & Ó Cinnéide, 2012; O'Keeffe & Cinnéide, 2006), testing (Ribeiro, Zenha-Rela, & Fernández de Vega, 2009; Wegener, Baresel, & Sthamer, 2001),cloud computing (Harman, Lakhotiaa, Singerb, Whiteb, & Yooa, 2012), software modularisation (Mitchell & Mancoridis, 2001a; Praditwong et al., 2011) and quality assurance (Khoshgoftaar, Khoshgoftaar, & Seliya, 2004).

A range of search techniques exist, including gradient methods, direct search or metaheuristics. Metaheuristics are more common in recent literature, because the high complexity of many problems does not suit gradient or direct search methods (Talbi, 2009). The term *heuristic* refers to problem-solving strategies that 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). Coello, Lamont and Van Veldhuisen (2007) refer to heuristic algorithms as 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 (Talbi, 2009). For example, Lin and Kernighan (1973) introduce 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). Correspondingly, the term 'meta' relates to 'upper' or 'higher level methodology'.

Metaheuristics can be classified into a range of different categories. Common classifications of metaheuristics into nature inspired vs. non-nature inspired, deterministic vs. stochastic, single-objective vs. multiple-objective, and single solution vs. population based algorithm can be found in the literature (Coello et al., 2007; Talbi,

2009). The metaheuristics applied within the present research are Multi-Objective Evolutionary Algorithms (*MOEA*s). Based on the classifications just noted *MOEA*s belong to the group of multi-objective, population based, stochastic and nature inspired algorithms.

### *2.4.1 Evolutionary Algorithms (EA)*

The objective of this section is to describe and discuss the main components of *Evolutionary Algorithms (EAs)* and to link these concepts with the targeted application in the domain of architecture reconstruction. A wide number of different variations of *EAs* exist e.g. *Genetic Algorithms*, *Particle Swarm Optimization*, *Scatter Search*, *Differential Evolution*, *Decomposition Based Evolutionary Algorithm*, *Bee Colony*, *Artificial Immune System*, and hybrid variations as mixed implementations of the former (Coello et al., 2007). Nevertheless, the common principles of *EAs* are similar. Hence, the description of EAs provided here is deliberately based on a unifying presentation of the common basics as the strategy adopted within this work is to apply and evaluate different kinds of *EAs*.

Coello et al. (2007) and Glover and Kochenberger (2003) both specify the essential components for the definition of an *EA* as a representation/encoding, a fitness function, a population, a parent selection mechanism, variation operators and a survivor selection mechanism.

#### *The Solution Representation*

Coello et al. (2007) and Talbi (2009) highlight the definition of a suitable representation as one of the fundamental components of the implementation of metaheuristics. A representation is a model of the original problem domain and is therefore an encoding of the original solution space into the *EA* solution space. Instances of the representation are the individuals that are generated by the *EA* implementation. Solutions of the original problem domain are called phenotypes and the encoded representations within the *EA* implementation are called genotypes (Coello

et al., 2007). Usually, a one-to-one relationship from genotype to phenotype is common but one-to-many and many-to-one relationships are also feasible. Defining another relationship than one-to-one has an impact on the size of the genotype and phenotype space and consequently the level of detail to which solutions can be identified. Within the present research a one-to-one mapping has been implemented. To apply an *EA* in a new problem domain a suitable encoding to present the genotype needs to be defined and a mapping function has to be implemented which transforms the genotype into the phenotype. The genotype consists of a number of decision variables (Agoston E Eiben & Smith, 2003). The decision variables are the fragments of the genotype, which are altered by the *EA* implementation. A decision variable features a type. Linear encodings of decision variables such as binary, discrete, and permutation encodings but also nonlinear encodings such as tree representations can be handled by *EAs* (Coello et al., 2007). Mixed encodings are supported in some *EA* implementations (Talbi, 2009). Hence, the encoding of the genotype with established decision variables makes *EAs* a universal and robust tool suitable for application in diverse problem domains. In comparison to classical optimisation methods the universal encodings prevent the necessity for adaption of the actual algorithm implementation to enable their application in new problem domains.

Figure 11 depicts a simple example of the encoding that has been developed in this research. The depicted encoding scenario enables the classification of low-level artefacts into a high-level conceptual architecture model.

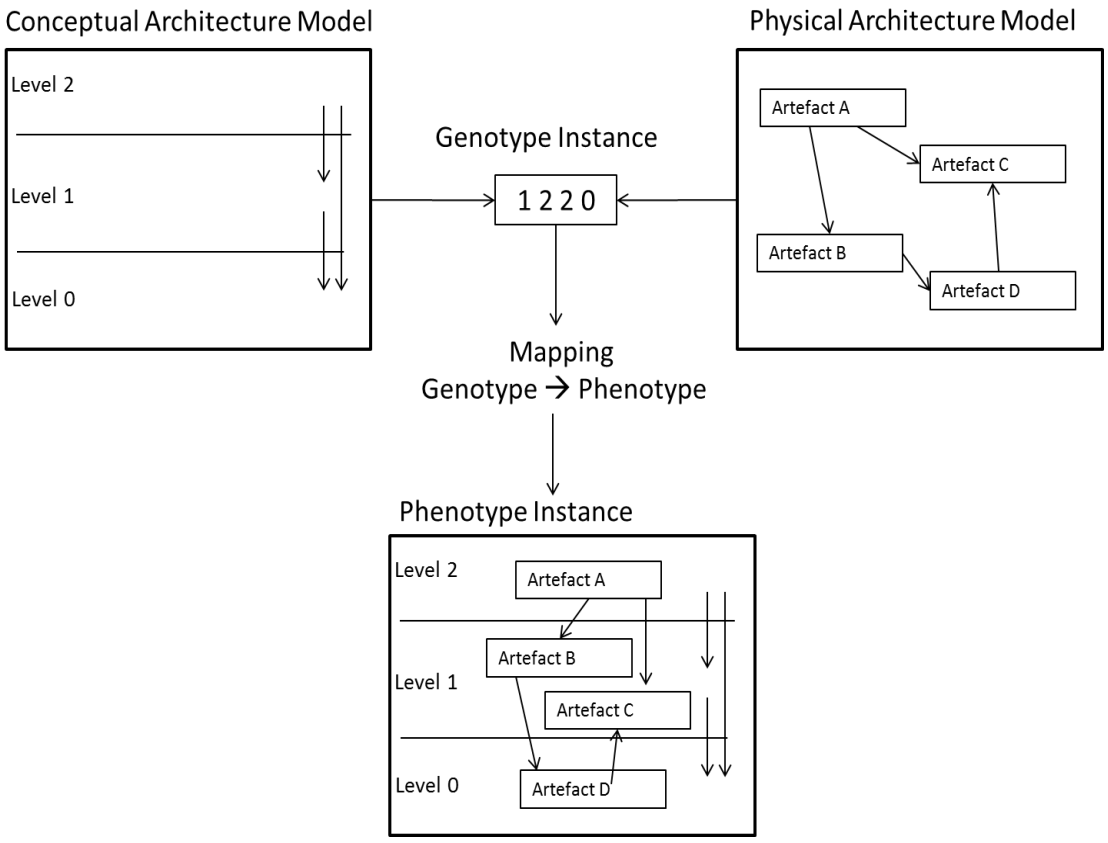


**Figure 11: Encoding of architecture classification problem**

Each of the low-level artefacts of the software system is represented by an integer decision variable. The range of the integer decision variables is defined by the number of high-level artefacts of the conceptual architecture model.

Specific genotype instances are created by the applied *EA* implementation. Hence, the genotype instance is a numerical representation of the classification of the artefacts of the system into the subsystem of the conceptual architecture model.

In the present research a representation is developed that enables the classification of low-level artefacts into high-level artefacts on different abstraction levels. Additionally, high level artefacts and dependencies can be discovered in the search process. Section 4.1.3 describes the representation that has been implemented in the present research in more detail.

Obviously, the genotype instance does not enable a direct statement to be made about the fitness of the classification. Consequently, the genotype representation is transformed into a phenotype instance that enables the assessment of the quality of a generated solution.

### *The Fitness Function*

Whilst *EAs* or metaheuristics in general can produce a variety of different solution candidates, the fitness of these solution instances needs to be determined to allow a ranking of the solutions. This requirement is satisfied by the fitness function (also referred to as the objective or evaluation function) (Coello et al., 2007; Agoston E Eiben & Smith, 2003; Talbi, 2009). The fitness function defines the quality of a solution in the specific problem context. Hence, a fitness function quantifies the optimality of a solution (Talbi, 2009) and expresses the goal of the search. The fitness function is usually composed from quality measures of the genotype space (Agoston E Eiben & Smith, 2003). As such, the quality of the fitness function depends on a sound orchestration and estimation of the influencing factors. Mostly, the fitness function is designed as a surjective transformation which maps the solution space into the set of real numbers $f: S \rightarrow R$. Consequently, the fitness function enables a ranking of solutions. Feasible fitness functions to evaluate the fitness of a generated solution in the targeted problem domain for this research are, for example, the number of architecture violations, the number of cycles, and the cohesion or the coupling on any artefact abstraction level. A detailed description of the software design metrics that have been selected as being relevant for the implementation of the present research and consequently can be employed as fitness functions is given in section 4.1.2.

### *The Evolutionary Algorithm Process*

The encoding of solutions and the fitness function to evaluate solutions represent the basic components for the representation and evaluation of *EAs*. However, the process of *EAs* is also of central importance to drive the evolution of solutions towards

their further improvement. Figure 12 illustrates the basic workflow of a general *EA* implementation.

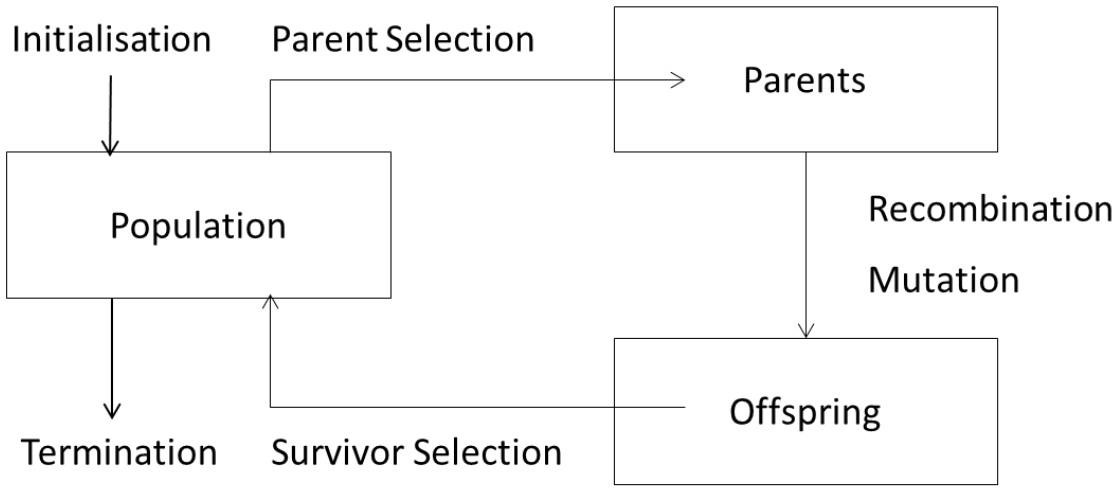


**Figure 12: Evolutionary algorithm process (adapted from Eiben-Smith (2003))**

A population is a set of genotypes and it forms the unit of evolution. Hence, *EA* implementations alter the population by changing decision variable values and adding or removing individuals (Coello et al., 2007). In the targeted problem domain the alteration of decision variable values translates to the reassigning of software artefacts into different conceptual architecture artefacts.

A tuning parameter, often required for the execution of an *EA* implementation, is the population size which defines the number of individuals within the population. Within the *EA* process an initial population of individuals is created. Talbi (2009) and Coello et al. (2007) list *Random Generation*, *Sequential Diversification*, *Parallel Diversification and Heuristic Initialisation* as possible strategies to create the initial population.

The individuals of the population are evaluated and solution aspects are selected to seed the next generation based on the evaluated fitness of the solutions. This process is called parent selection (Agoston E Eiben & Smith, 2003) or mating selection (Coello et al., 2007). If an individual is selected this individual becomes a parent for the next

generation. The parent selection process within *EA* implementation is usually based on stochastic methods. For example, the selection process prefers individuals with higher fitness as parents of the next generation, but also a weaker individual might have a chance to reproduce (Talbi, 2009). This behaviour allows *EAs* to escape local minima by preventing the stagnation of the population.

Variation operators such as recombination and/or mutation strategies are applied to the parents to create the next population (Agoston E Eiben & Smith, 2003). Recombination conglomerates aspects of two or more individuals (parents) and results in one or more individuals (children).

The selection of attributes from the parent individuals is usually probability based and depends on random drawings. However, often recombination tuning parameters can be injected in the EA process to influence the recombination method. The principle of recombination is inspired by nature and assumes that creating offspring from individuals with desirable features can lead to individuals which combine these desired features and outperform the parents (Luke, 2010). However, this evolutionary process does not work in every case and might lead to offspring which are no better or worse than the parents (Luke, 2010). Hence, within the *EA* process multiple instances of offspring are created and evaluated in the hope of finding an improved solution. Crossover is often used as a synonym for the term recombination (Talbi, 2009).

The importance of recombination in the different *EA* dialects varies. For example, Glover and Kochenberger (2003) state that within *Genetic Algorithms* and *Genetic Programming* recombination is often the only applied variation parameter. In *Evolutionary Programming,* on the other hand, recombination is almost never used as the probability based mating of programme representations rarely produces feasible and improved solutions (Glover & Kochenberger, 2003). Nevertheless, within the targeted problem domain the application of recombination concepts is probably of high relevance due to the existence of interdependencies between the artefacts of

architecture configurations. Hence, recombination strategies might be able to capture such solution aspects and recombine solutions in a meaningful way to quickly enable an improved solution development.

A mutation is a unary operator changing aspects of one or more individuals of a population (Coello et al., 2007; Glover & Kochenberger, 2003). The application of a mutation is probability based. Technically, this means randomly changing one or more decision variable values. Again, mutation has different relevance within the different *EA* dialects. For example, *Genetic Algorithms* and *Genetic Programming* abdicate the application of mutations almost completely (Glover & Kochenberger, 2003). Nevertheless, mutation strategies might indeed be of value for application in the targeted problem domain. The capability to evolve a population by applying only recombination techniques has limitations due to the explicit reliance on previously visited solution attributes (Andrews, 2006). Hence, the application of mutation operators is promising to drive exploration into new areas of the search space if a further improvement of solutions based on the recombination of solution aspects is infeasible. An example of such a scenario in a software modularisation problem is the assignment of artefacts which have only been conducted in some of the conceptual architecture artefacts. Hence, some classification aspects cannot be explored through the explicit application of recombination strategies.

Variation operator implementation features usually a set of tuning parameters to determine aspects such as application probability or diversity. The tuning of these parameters has a potential impact on the performance of the *EA* implementations. Agoston E. Eiben, Michalewicz, Schoenauer, and Smith (2007) state that the ideal configuration of tuning parameters depends on the nature of the problem and the type, number and range of the decision variables that are utilised to represent the problem. Furthermore, Agoston E. Eiben et al. (2007) describe that the parameters of *EA* implementations are not necessarily independent. Evaluating all feasible parameter

combinations in a systematic manner is practically impossible and the process of parameter tuning can be time consuming even if parameters are tuned individually.

A widely applied practise to identify useful parameter configuration is the iterative tuning of parameters until acceptable results are found that enable thorough conclusions on the feasibility of an approach.

For example, the closely related research presented in Praditwong et al. (2011) applies a *GA* implementation in a software modularisation context which implements a single-point-crossover and single-point-mutation operator. Praditwong et al. (2011) identified a metaheuristic tuning during such an iterative process. They applied a metaheuristic configuration that features a population size of 10N and a maximum number of generations of 200N, where N is the number of low level modules of the analysed system. Single-point crossover and single-point mutation are applied as mutation operators. The probability of crossover is 0.8 if the population size is less than 100 and 1.0 if the population size is higher. The probability of the mutation operator is 0.004 log2(N). Barros (2012) applied an identical parameter tuning in their study. Praditwong et al. (2011) and Barros (2012) state that the applied configuration is suitable to demonstrate the feasibility of their approach but do not claim that the utilised parameter tuning is an optimal configuration for the developed software module classification problem.

In summary, the support of recombination and mutation strategies is important for the thorough implementation of *EAs* in a software modularisation and architecture reconstruction context. However, previous research has not shown to which extent recombination and mutation operators are involved in identifying promising solution candidates. Hence, the approach implemented in this research should feature flexible support to employ recombination and mutation strategies to enable the evaluation of the relevance of recombination and mutation strategies within the software modularisation and architecture reconstruction process.

The execution of recombination and mutation leads to the creation of a set of new individuals (the offspring). The offspring compete with the individuals of the old generation for a place in the next generation. This last step of the EA iteration is called survivor selection or replacement (Coello et al., 2007; Agoston E Eiben & Smith, 2003; Talbi, 2009). The replacement phase does not vary much from the parent selection process apart from its occurrence after offspring creation. As discussed earlier, the population size is generally (apart from some *EA* dialects) a constant during the runtime. Naturally, the number of offspring individuals exceeds the defined population size. Hence, it has to be decided which individuals survive and are a part of the next generation (Glover & Kochenberger, 2003). This decision is usually based on the fitness ranking of the offspring and the individuals in the previous population (Coello et al., 2007; Talbi, 2009).

Another variable to determine the survival of solutions might be the age of the population (Coello et al., 2007). Survival selection is, compared to the stochastic selection strategies of the *EA* process, usually deterministic (Glover & Kochenberger, 2003).

The application of *EA* entails two key concepts, first the application of variation operators to create diversity and novelty, and second the selection strategies to create environmental pressure to drive the search towards better fitness of population individuals (survival of the fittest). This combination of variation and selection operators leads to a fitness improvement in consecutive populations. This process of parent selection, recombination, mutation and survivor selection is repeated until a termination criterion is fulfilled. Two scenarios of termination are generally applied within the optimisation discipline. First the termination can be based on a desired fitness of the optimal solution. The search is terminated once that level of fitness has been reached. However, this requires that the optimisation algorithm is able to discover a solution that features this level of fitness and that the desired level of fitness can be specified. The other and more commonly applied strategy in the domain of *EA* is based

on the definition of one or more stopping criteria. Common scenarios for this are the definition of maximum elapsed CPU time or the maximum number of fitness evaluations (Luke, 2010). The utilisation of CPU time is probably not desired in a research project that is reliant on the empirical evaluation of search outputs. One reason for this is that execution on different machines would lead to different search outputs as the performance of the evaluation machines might not necessarily be identical. Such a scenario would bias the comparability of search results. Hence, the reliance on a fixed number of iterations or generations is likely to be a better stopping criterion in the present research to enable the rigorous comparability of search results.

A desired execution of an *EA* features fast convergence with good diversity of evaluated solutions within the solution space. The survivor selection operator rewards improvement in the objective space and consequently drives the *EA* towards optimal convergence. The crossover and mutation operators, on the other hand, seek to obtain diversity in the objective space. Hence, the survivor selection mechanism and the crossover and mutation operators are in an on-going competition for convergence and diversity.

Besides the tuning of the algorithm parameters, the performance of *EA* implementations is also affected by characteristics of the problem. However, both Coello et al. (2007) and Talbi (2009) state that evolutionary algorithms have been proven to be robust and successfully applied in a wide and diverse range of difficult problem domains such as problems with high-dimensional decision variable and objective spaces, problems which require the search to travel through large infeasible spaces before finding a feasible space, and problems with many local optima.

State of the art *EA* implementations can apply strategies to leverage the impact of decision variables on the solution quality by implementing a dynamic exploration of the search space. Depending on the evolutionary algorithm, different strategies are applied to exploit the structure and relationship of the decision variable space. Correspondingly,

Coello et al. (2007) state that evolutionary algorithms do not only attempt to converge towards an optimal fitness value from one generation to the next, in fact the application of evolutionary algorithms is also a process of adaption in which the fitness of an individual is not only seen as an objective function but also as an expression of environmental requirements.

The motivation of this work is to find structures within the source code of a software system and classify these into conceptual architecture models that can be utilised as development blueprints in later development phase. Hence, *EAs* might deliver a potent method to identify useful structures for the reconstruction of software architectures.

### *2.4.2 Multi-Objective Evolutionary Algorithms (MOEAs)*

The present research applies *Multi-Objective-Evolutionary-Algorithms (MOEAs)* in the domain of software architecture reconstruction. The previous section illustrated the basic principles of *EAs* and their potential for implementation in the domain of software architecture reconstruction. Nevertheless, section 2.2 highlighted that the design of a software architecture is driven by multiple objectives. For instance, a potential conflicting goal within the present research is the minimisation of cycles and the maximising of cohesion within software artefacts. Both of these goals are valid from a software design perspective. Exclusively focusing on the minimisation of the number of cycles or only focusing on maximising the cohesion within modules will likely create solutions that are biased towards one objective and feature unacceptable performance in the other. This section illustrates the reasoning for the application of *MOEA*s within the present research and the basic concepts that underlie *MOEA*s.

In single objective optimization, a metaheuristic will converge towards one solution that features optimality based on the definition of one objective (Glover & Kochenberger, 2003). The outcome of a single objective search is one solution that outperforms every other solution that has been visited during the search process in terms of the defined objective. Any other enhancing or degrading solution attribute is

not considered during the selection process. This unconditional optimisation of one single attribute of the solution, without allowing any trade-off in other solution attributes of the solutions, will hardly offer a sufficient solution for non-trivial problems.

The incorporation of multiple objectives might then provide a more balanced solution. The goal of multi-objective optimisation is to find a solution that trades off multiple objectives. The search process itself is then a battle between the individual objectives (Coello et al., 2007). Research in the operations research discipline suggests different approaches to including multiple objectives into the optimisation process.

Eiben and Smith (2003) highlight the combination of objective values into a single value as one potential approach to include multiple objective measurements into a search. Under such an approach single objective optimisation techniques can be utilised after the objective values have been aggregated into a single value. Advanced approaches suggest the definition of a weighting scheme to enable an importance ranking between the objectives (Agoston E Eiben & Smith, 2003). This may not be straightforward, however, as the application of a weighting scheme implies that the relevance of objectives can be established before the conduct of the optimisation. Additionally, different weighting schemes with the same objective setting will discover different final solutions. Moreover, only one single solution is found and no alternatives are discovered during the search. Additionally, different objectives might feature a different range of values. One objective might feature values in the range 0 - 1.0 and another objective might feature values 0 - ∞. A normalisation is not feasible before the search is completed unless the absolute range of objective values is known. In short, the application of weighting schemes to rank objective goals is complex and hence not used in this research.

Another approach is to incorporate all objectives as separate functions. The principle of *Non-Dominance* introduced by Edgeworth (1881) and extended by Pareto (1896) is often utilised instead of optimizing all the objective functions individually. A

solution is dominated if, within a set of solutions, at least one other solution exists that outperforms it in all objective values. A *Non-Dominated* or *Pareto-Optimal* solution outperforms every member of a set of solutions in at least one objective but performs worse or equal in at least one other objective (Coello et al., 2007). In other words, a solution is *Non-Dominated* or *Pareto-Optimal* if there is no known solution that outperforms at least one objective without decreasing the performance in any other objective. It is important to note that the objectives in which a solution outperforms, is equal to or is worse than another solution can differ from solution to solution. A set of solutions features *Pareto-Optimality* if every solution of the set does not dominate any other solution of the set. Given that no single solution dominates all solutions of an optimisation run, the result of a multi-objective optimisation is a set of *Non-Dominating* solutions. The set of all *Non-Dominated* solutions of an optimisation run is called the *Pareto-Front*. Each of the solutions of the *Non-Dominated Pareto-Front* dominates an area of the search space. Figure 13 illustrates a *Pareto-Front* with three solutions *A, B, C* and the corresponding dominated solution space.

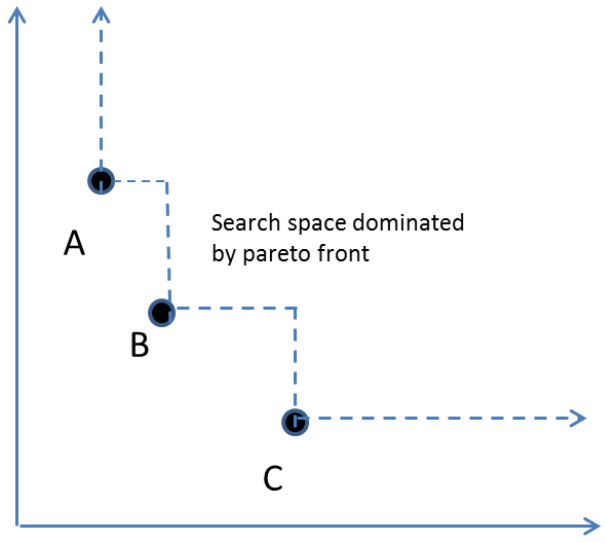


**Figure 13: A non-dominated *Pareto-Front***

The disadvantage of the application of the *Non-Dominance* principle within metaheuristics is that *Pareto-Fronts* cannot be ranked directly. The assessment of the

performance of an optimisation run that features the *Non-Dominance* principle is more complex than the performance evaluation of a single objective optimisation. However, performance indicators exist to evaluate the performance of *MOEAs*. A detailed overview of multi-objective performance indicators that are applied in the present research is presented in section 3.3.4.

Additionally, the *Pareto-Front* contains solutions that excel in some objectives but may perform poorly in others. In other words, it cannot be determined which solution of a *Pareto-Front* is the best from the perspective of a stakeholder. The understanding is that only limited approaches exist to effectively review *Pareto-Fronts* (Blasco, Herrero, Sanchis, & Martínez, 2008; Kasprzyk, Reed, Characklis, & Kirsch, 2012).

Sayyad and Ammar (2013) review a total of 51 papers that applied multi-objective approaches in the *SBSE* discipline between 2004 and 2013 and classify the papers based on the application area in the categories requirements (10 papers), design tools and techniques (15 papers), testing/debugging (16 papers) and software project management (10 papers). According to this classification the present research would be placed in the category of design tools and techniques. However, only the research of Praditwong et al. (2011) and Barros (2012) is directly related to the objectives of the present research, as these studies apply multi-objective approaches in the area of software modularisation. A detailed description of these two research contributions is given in section 2.4.3. Furthermore, Sayyad and Ammar (2013) analyse the papers based on the number of applied objectives, the applied multi-objective search algorithms, the application of an open source multi-objective framework and the application of quality indicators. Figure 14 depicts a classification of the number of objectives and the corresponding research papers identified by Sayyad and Ammar (2013).

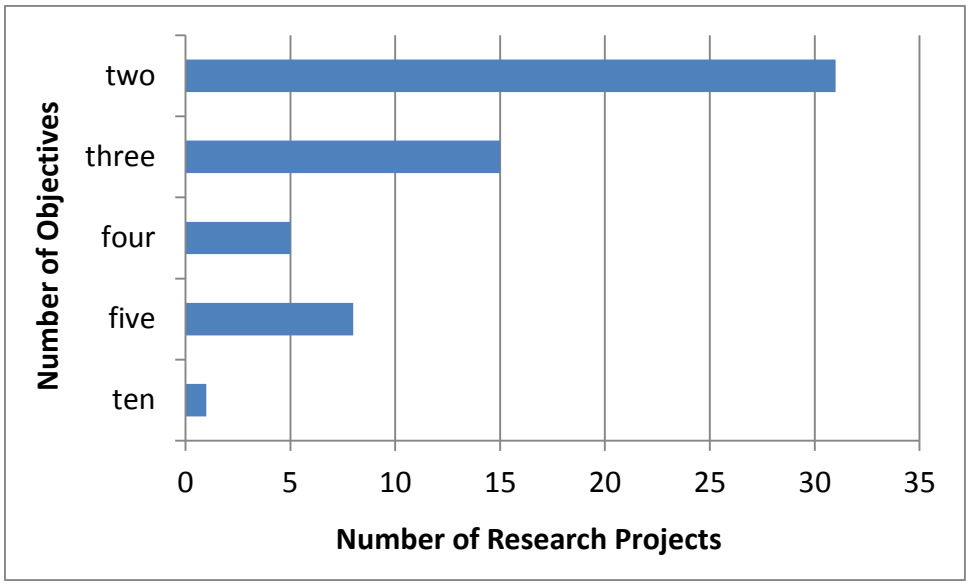


**Figure 14: Number of objectives in multi-objective research**

Only a few of the research efforts consider more than three objectives. However, it might be questioned whether real world applications can be simplified to such a small number of objectives and still be a realistic model of the targeted problem domain. Seven of the reviewed papers present different formulations for a specific problem by applying different objective settings for a problem presentation. Praditwong et al. (2011) and Barros (2012) both use two fitness functions with 4 or 5 objectives. Barros (2012) also compares the efficiency of the two different objective settings with each other (compare: section 2.4.3). However, it can be stated, based on all the reviewed papers by Sayyad and Ammar (2013), that none of the approaches allows stakeholders to define or change the objective configuration of the fitness function.

Figure 15 gives a compilation of the frequency of the application of specific algorithms as presented by Sayyad and Ammar (2013).

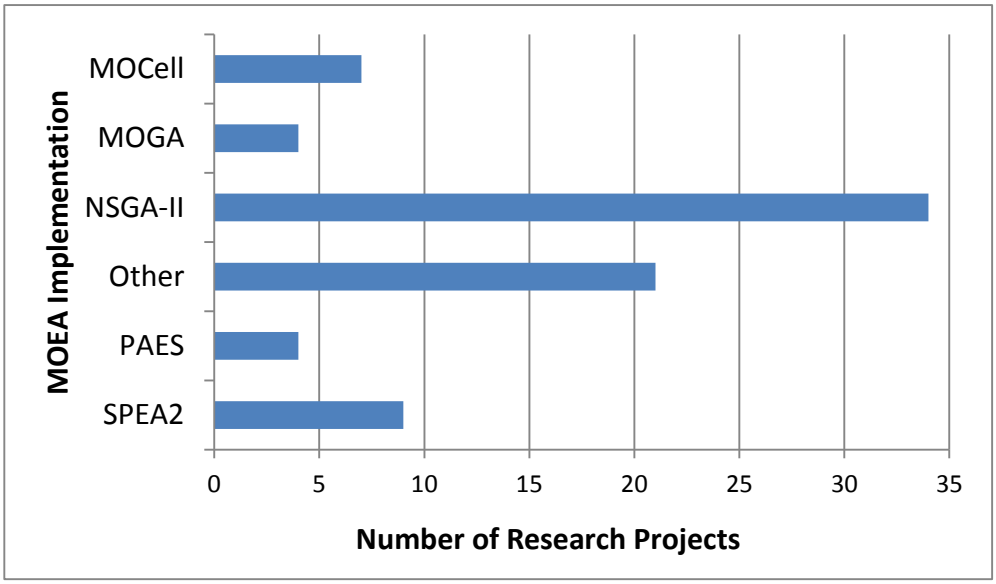


**Figure 15: Frequency of application of multi-objective algorithms**

NSGA-II, introduced by Deb, Pratap, Agarwal, and Meyarivan (2002), is the algorithm of choice in 53% of the reviewed papers. Additionally, 17 papers (33%) reported using implementations of the algorithms that are available in tools such as *jMetal*[12] (13 papers), *Matlab* (2 papers), *Frontier* (1 paper) and *Opt4J*[13] (1 paper). The researchers implement proprietary algorithms in the remaining two thirds of the papers. Sayyad and Ammar (2013) report that 36 papers (70%) used only a single algorithm whereas 15 papers (30%) used multiple algorithms for comparison purposes. Praditwong et al. (2011) used a self-implemented two-archive *GA* implementation and Barros (2012) utilised the NSGA-II implementation from the *jMetal* framework. 15 papers (30%) used quality indicators to assess the quality of the *Pareto-Front*s. *Hypervolume* (HV) was the most widely used indicator and was applied in 12 papers. A detailed overview of

[12] http://jmetal.sourceforge.net/
[13] opt4j.sourceforge.net/

optimisation performance metrics and reasoning for the application of specific optimisation performance metrics in the present research is given in section 3.3.4. Praditwong et al. (2011) use no *Pareto-Front* quality indicator; instead they utilise the *MQ* measure introduced by Mitchell and Mancoridis (2001a) to compare the performance with a single objective approach. Barros (2012) utilises *GenerationalDistance* and *Error Ratio* as performance indicators. The review of the literature presented in Sayyad and Ammar (2013) reveals some methods that should be followed for the implementation of a multi-objective architecture reconstruction and classification approach.

In summary, current research has neglected to compare the performance of different *MOEA* implementations and *MOEA* tunings to enable conclusive statements to be made regarding the performance of different algorithm implementations. Additionally, the application of *Pareto-Front* quality indicators, different problem formulations and objective settings are often overlooked in multi-objective *SBSE* research. This research closes this research gap by firstly offering the flexible employment of a variety of *MOEA* implementations in different architecture reconstruction scenarios and secondly by providing a multi-objective evaluation framework that enables the tool-driven comparative evaluation of multi-objective solution sets.

### *2.4.3 Search Based Modularisation Approaches*

Within the present research an approach has been developed that applies *MOEA* implementations in the area of architecture reconstruction. This approach is implemented as a software modularisation approach that considers a conceptual architecture model during the modularisation process. In the reviewed literature no approach describes the consideration of a conceptual architecture model within a search based modularisation approach. Nevertheless, various approaches implement search based techniques within low-level software modularisation. The review of such related approaches is very useful to inform the selection of methods and techniques for

possible application in the challenging field of software architecture reconstruction. This section presents a summary of the research that applies search based software engineering in the problem domain of software modularisation, decomposition and partitioning. In particular, Mancoridis, Mitchell, Chen and Gansner (1999) and Seng, Bauer, Biehl and Pache (2005) present the main approaches to applying *SBSE* techniques to reengineer the structure of a software system.

Mancoridis et al. (1999) show that the structure and complexity of cluster analysis as applied to software systems mean that search based software engineering is a promising approach to create feasible solutions. The objective of the approach, first presented in Mancoridis et al. (1999) and then extended in Mitchell (2002), Mitchell and Mancoridis (2006) and Mitchell and Mancoridis (2008), is to discover a cluster configuration which features high cohesion within clusters and low coupling between clusters. The input data is a dependency graph representing a software system with software artefacts and their dependencies. The output is a so-called *Module Dependency Graph (MDG)* which features a classification of the nodes into a number of clusters. The number of modules is a result of the search process and cannot be controlled externally. The search utilises an iterative single-solution based approach that reassigns nodes into modules. The instances of the *MDG* that are visited during the search are evaluated utilising the *Module Quality (MQ)* measure as an objective function (Anquetil & Lethbridge, 1999). The *MQ* measure is designed to reward high cohesion in modules and penalise high coupling between modules. The *MQ* measure for a *MDG* with *n* modules is calculated by computing a *Cluster Factor (CF)* for each of the *n* clusters of the *MDG*. The *CF* is defined as a normalized ratio between the total weight of the internal edges and half of the total weight of the external edges. The weight of an edge can be defined as a numeric value as part of the input dependency graph or, if no value is given, the default edge weight is 1.0. Anquetil and Lethbridge (1999) define the *CF* for a module $m$ with a set of inter-module edges ($\mu i$) and a set of intra-module edges ($\varepsilon\ i, j$ or $\varepsilon\ j, i$) as depicted in the following equation:

$$\text{CF } m = \begin{cases} 0 & \mu i = 0 \\ \dfrac{2\mu i}{2\mu i + \sum_{\substack{j=1 \\ j \neq i}}^{k} (\varepsilon i,j + \varepsilon j,i)} & \end{cases}$$

The weighting of an edge determines the strength of an edge connection. The weighting is considered during the calculation of the *CF* metric and correspondingly has an impact on the search output. However, as illustrated in section 2.4.2 the definition of a weighting scheme is itself a complex task and might prevent a search from discovering promising solutions. Hence, an unbiased search as applied in a *Non-Dominance* driven multi-objective strategy, which navigates through the search space regardless of pre-defined user weightings in conjunction with a subsequent containment of the solution space, is a more unbiased and flexible solution approach.

Mitchell and Mancoridis (2006) depict the *MQ* measure to determine the absolute quality of a *MDG*. The *MQ* measure is calculated as the sum of all clusters of a *MDG*. Hence, the formula to calculate the *MQ* measure for a *MDG* with $m$ modules is defined as follows $MQ = \sum_{i=1}^{m} CF\ i$. The *MQ* measure operates as a fitness surrogate to express the quality of a solution in a single value. The feasibility of the visited solutions is based on the ranking of their *MQ* value in relation to the *MQ* value of other solutions.

The approach developed by Mancoridis et al. (1999) is, for example, implemented within the *Bunch* artefact. Correspondingly, *Bunch* is applicable to identify modularisation compositions that feature low coupling and high cohesion based on a given dependency graph.

The reengineering of module constellations in software systems based on low numbers of efferent/afferent edges and a high degree of internal edges is certainly a valid approach. However, contemporary considerations of coupling within the software engineering literature do not assert that the dependencies between two modules should be reduced to an absolute minimum. For example, it is not necessarily a bad

design if an artefact on any abstraction level uses other artefacts heavily. However, the work of Mancoridis et al. (1999) defines loose coupling as a low number of dependencies between subsystems. Correspondingly, other design preferences that could operate as feasible objectives to reengineer modular software architecture designs are, for example:

- A desired (low) number of cyclic dependencies between artefacts
- A desired number and structure of conceptual architecture artefacts
- The differentiation between valid and invalid dependencies between conceptual architecture artefacts
- A maximum number of efferent or afferent coupled artefacts to control responsibility and independence of artefacts
- The mutual grouping of abstract classes and interfaces with the corresponding implementers.

The *Bunch* tool supports a hill-climbing algorithm, an exhaustive clustering algorithm, and a *Genetic Algorithm (GA)* as well as a *Simulated Annealing (SA)* implementation (Mancoridis et al., 1999; Mitchell, 2002; Mitchell & Mancoridis, 2008). The implemented algorithms are single objective optimisation algorithms. The studies demonstrate that the complexity of the clustering problem makes the application of the exhaustive search infeasible for non-trivial dependency graphs (Mitchell & Mancoridis, 2006). Mitchell and Mancoridis (2002) and Mahdavi et al. (2003) state that the performance of their hill-climbing implementation out-performs the *GA* and *SA* implementations based on *MQ*–objective achievement.

The hill-climbing clustering algorithm implemented within *Bunch* starts with a random cluster configuration of the *MDG* as the current search representation. A node from the cluster configuration is selected and reassigned to another cluster with the objective to find cluster configurations that feature an improved *MQ*. The solution is evaluated based on its *MQ* value. The solution becomes the new current representation

in the search space if it features better fitness based on the *MQ* measure. This process is iterated until an improvement is found. The hill-climbing implementation stops when no configuration in the direct neighbourhood can be found with a higher *MQ*.

The hill-climbing clustering algorithm implemented in the work of Mancoridis et al. (1999); Mitchell (2002); Mitchell and Mancoridis (2001a, 2006, 2008) is initialised with randomly created initial cluster configurations. Hence, it is unlikely that two clustering runs will produce identical clustering results. Mitchell and Mancoridis (2002; 2001a) note that stakeholders would likely expect similar clustering outputs. Mitchell and Mancoridis (2001a) introduced *EdgeSim* and *MeCI* as cluster similarity measurements that, besides the placement of modules into clusters, also take the connectivity of the system modules into account. For each edge of the module dependency graph, *EdgeSim* measures the number of times that an edge serves as an intra-edge across multiple clustering runs. The intra-edge count is divided by the total number of runs to attain a normalised intra-edge count that is independent of the total number of clustering runs. The frequency of the intra-edge count is aggregated into ranges. To enable the comparison of the *EdgeSim* measure across systems the frequencies are normalised by dividing the number of edges of the frequencies with the total number of edges of the clustered module dependency graph. Mitchell and Mancoridis (2001a) argue that a high degree of edges within the high percentage ranges and the low percentage ranges is favourable. A high count of edges within the high percentage ranges indicates that a high number of edges have repeatedly served as intra-module edges. A high edge count within the low percentage range indicates that a high number of edges have repeatedly been inter-module edges. Hence, a high percentage in the low number of inter-edge counts and in the high number of inter-edge counts indicate good stability or reliability in a clustering algorithm. A high number of edge counts within the mid-percentage ranges show that the algorithm produced solutions in which dependency edges have served as an inter- or intra-module edge a balanced number of times, which is not a desirable output in terms of stability.

Mitchell and Mancoridis (2008) evaluate the stability of their implemented hill-climbing algorithm against a set of seven real systems and six randomly generated dependency graphs. The systems varied in size between 28 and 558 nodes and between 85 and 3793 node edges. Every system is clustered a hundred times and attributes of the clustering results, such as cluster count and *MQ,* are compared in scatter plots. The authors argue that agglomeration within the scatter plots can be observed, which supports the claim that there is similarity in certain attributes of the solutions. Additionally, the *EdgeSim* metric is calculated with frequency ranges of zero (0%), low (0.1% - 10%), medium (10.1% – 75%), and high (75.1% – 100%). In spite of the wide range of the medium frequency bin, only up to 25% of the edges of the 13 systems under evaluation are within the medium range. This indicates fairly good stability of the evaluated hill-climbing algorithm as at least 75% of the edges are within the zero, low and high ranges. Additionally, the evaluated hill-climbing algorithm performs better with dependency graphs derived from real systems than with randomly generated dependency graphs based on the *EdgeSim* metric.

More importantly, the fact that Mitchell and Mancoridis (2001a, 2008) differentiate between the reliable assignment of inter- and intra-edges raises the idea that not only is the objective to optimise the absolute clustering quality based on cohesion and coupling important, but that aspects such as which edges are inter- or intra-edges and overall dependency relationships should be considered during the modularisation process.

A general issue that can also be encountered in the *Bunch* implementation is that hill-climbing implementations stall in local optima (Glover & Kochenberger, 2003). Mahdavi et al. (2003) propose a multi-hill-climbing approach to overcome the problem of premature convergence of single hill-climbers towards local optima. The approach that is presented in Mahdavi et al. (2003) is divided into the execution of an initial set of hill-climbs and a following set of best hill-climbs that are identified to a cut off threshold. Common features of the best solutions of the initial hill-climbs are identified. These common features form building blocks for a set of subsequent hill-climbs. A

building block features a set of nodes that are tied in to a particular cluster if all initial hill-climbs place the nodes under consideration in the same cluster. The research of Mahdavi et al. (2003) is motivated by the hypothesis that the compliant allocations of the initial hill-climbing runs deliver evidence that good solutions should also contain these node allocations in common clusters. In Mahdavi et al. (2003) a set of twenty-three initial hill-climbs are executed. After the initial run the building blocks are calculated and another set of twenty-three hill-climbs are performed using the calculated building blocks as fixed input sets. The result sets of the final runs are merged and the best solution of all runs is presented as the final solution. The *MQ* measure is utilised as a fitness function to determine the fitness of the visited solutions. Ten initial runs, each with a different cut off threshold starting from the best ten percent to the best one hundred percent (effectively no cut off), are executed to reveal the most promising cut-off point. The study has been conducted with nineteen dependency graphs extracted from small systems with twenty modules to larger systems with just over four hundred nodes.

Mahdavi et al. (2003) find that the application of a multiple hill-climbing technique together with the proposed building block technique is able to find better solutions than a pure single hill-climbing run based on the absolute *MQ* measure. Threshold cut-off points of ten and twenty per cent reveal an improvement in terms of the *MQ* measure for all systems in comparison to a single hill-climbing run in all evaluated systems. Additionally, higher cut-off points also improved the final result within larger systems. Nevertheless, the approach suggested in Mahdavi et al. (2003) to overcome the stall of the search in local optima is rather complex. A more straightforward approach might be the application of *Evolutionary Algorithms (EA)* as such algorithm implementations are not as prone to stall in local optima.

Harman, Swift, and Mahdavi (2005) evaluate the robustness of the *MQ* and EValuation Metric (*EVM*) fitness function. The *EVM* is introduced by Tucker, Swift, and Liu (2001) and applies to problems of time-series data and clustering of gene expression

data. However, Harman et al. (2005) suggest that *EVM* is also suitable for modularisation problems as it rewards larger numbers of intra-module relationships, but does not penalize the occurrence of inter-module relationships as strictly as the *MQ* function. Even if not stated explicitly, research suggests that a less strict penalization of inter-module edges, and with that the minimisation of the absolute number of inter-module dependencies, is not necessarily an ideal approach to rebuild software system structures.

The *EVM* metric considers all possible relationships within a cluster and rewards those that actually exist within the *MDG* and penalises dependencies that do not exist within the *MDG*. To calculate the *EVM* metric every possible intra-module dependency of every cluster is visited. If the dependency exists within the dependency graph under consideration, the EVM is incremented by one. If, on the other hand, the dependency does not exist within the actual dependency graph, the *EVM* score is decremented by one. Tucker et al. (2001) and Harman et al. (2005) argue that this approach indirectly penalizes high coupling because re-arranging nodes between clusters can change high coupling between two modules to lower coupling between them and can also increase the *EVM* score of the solution.

It needs to be noted that the *EVM* metric rewards solutions with a high occurrence of cyclic dependencies within clusters. This might not be such an issue if the clustered nodes are compilation units, because the approaches under consideration do not feature functionality to resolve these anyway. However, if the modularisation is to be applied at higher abstraction levels, such as assignment of packages into subsystems, the occurrence of cyclic dependencies within subsystems and between packages within subsystems is unfavourable from a software design perspective (Fowler, 2001; Lakos, 1996; Martin, 2000).

Harman et al. (2005) evaluate the robustness of software modularisation using *SBSE*. They introduce artificial noise by mutating the occurrence of edges within the

dependency graph. The study applies the hill-climbing algorithm introduced by Mitchell and Mancoridis (2001a). Real and artificially generated systems with 20 to 174 nodes and 57 to 360 edges are evaluated. For each system twenty hill-climbing solutions are generated with the *MQ* and *EVM* fitness function, equally split between those with and without noise introduction. The run without noise introduction operates as the ideal clustering solution. The *Weighted–Kappa* (*WK*) metric, introduced in Altmann (1991), is utilised to evaluate the impact of the noise (edge mutations) on the performance of the two fitness functions. The *WK* metric expresses the similarity of two clustering results as a normalised value. Hence, the *WK* metric for each fitness function is calculated by using the noisy and noise-free cluster configuration from each fitness function. As expected, the study reports a decrease in the *WK* metric with a higher degree of noise for the *EVM* as well as the *MQ* clustering solutions. However, the *WK* similarity measured in the *MQ* clusters declines more substantially in comparison to the *WK* similarity measured in the *EVM* clustering solutions with an increasing level of noise. These findings indicate that the *EVM* metric features greater robustness when applied to real software systems (Harman et al., 2005). The introduction of noise can be considered as consistent with the erosion of a software system in which more and more arbitrary dependencies occur in conflict with the original architecture design of the system. It can be argued that the *EVM* metric copes more effectively with eroded systems due to the less stringent penalization of inter-module relationships and hence performs better in systems that feature a certain level of dependency erosion. This aligns with the previously proposed argument of the present research which states that minimisation of the number of inter-module dependencies is not necessarily the best objective to modularise software artefacts into conceptual architecture models. This is based on the reasoning that the erosion evident in a system should not bias the outcome of the modularisation as these unwanted dependencies do not reflect the conceptual architecture design of the system. Additionally, occurrences of dependencies between conceptual architecture artefacts are not necessarily undesirable. The occurrence of dependencies between conceptual architecture artefacts is legitimate as long as these dependencies follow the

aspired dependency design of the conceptual architecture model. Hence, other objectives need to be identified that rely on different software design aspects to rebuild the classification of software artefacts into the conceptual architecture and counter the explicit reliance on high cohesion and low coupling within modules.

Abdeen et al. (2009) apply *Simulated Annealing* to optimise class partitioning within the existing package structure of a software system. Measures for *Inter-Package Dependencies*, *Inter-Package Connections*, *Inter-Package Cyclic-Dependencies*, *Package Cohesion*, *Package Coupling* and *Package Cyclic-Dependencies* are agglomerated into a single objective fitness function. The fitness function of Abdeen et al. (2009) is limited to identify direct cycles. Hence, cycles that span across more than two elements cannot be identified with the implemented approach. However, the consideration of longer cycles is highlighted as being worthwhile as long cycles have an even stronger negative impact on the design quality of software systems (Fowler, 2001; Oyetoyan et al., 2013). Furthermore, the approach presented by Abdeen et. al. (2009) supports constraints that relate to package size, to the number of classes that are allowed to change their packages, and to specific classes and packages that are not eligible to be moved or changed. These constraints can be defined by development stakeholders before the execution of the optimisation process.

Abdeen et al. (2009) evaluate their approach in a set of four case studies. The evaluated systems *JEdit*[14], *ArgoUML*[15] , *JBoss*[16], *Azureus*[17] comprise between 812 and 4212 classes and between 19 and 380 packages. Thus, the evaluated systems are considerably bigger than the systems evaluated in comparable studies (Mitchell and Mancoridis, 2008; Praditwong et. al., 2011). Abdeen et al. (2009) execute the algorithm ten times for each software application due to the non-deterministic characteristic of

[14] *http://www.jedit.org/*
[15] http://argouml.tigris.org/
[16] www.jboss.org/
[17] http://sourceforge.net/projects/azureus/

the *Simulated Annealing* implementation. Each execution of the algorithm conducts a total of 1500 transformations of the package partitioning of the system. Abdeen et al. (2009) compare the average gain of the ten algorithm executions to the original package partitioning of the evaluated software applications for each of the six fitness function components. Hence, the optimisation is driven by the agglomerated single objective fitness function while the evaluation considers the individual objectives separately.

Abdeen et al. (2009) claim that the observed reduction of the average package coupling and the number of cycles during the case studies is an indication that a relatively small number of transformations can lead to a noticeable improvement in cyclicality and coupling within the evaluated systems.

Seng et al. (2005) also present a single objective approach that searches for an optimal subsystem decomposition by optimizing metrics and heuristics of good subsystem design. The approach of Seng et al. (2005) groups compilation units into a higher abstraction level. From a software design perspective the subsystems can be understood as packages or folders of the software system. Seng et al. (2005) apply a genetic algorithm implementation. The fitness function, in which *w1*, *w2*, *w3, w4* and *w5* are user defined weights, is defined as follows:

$$f = w1 \times cohesion + w2 \times coupling + w3 \times complexity + w4 \times cycles + w5 \times bottlenecks$$

The fitness function is based on the two most commonly adapted quality concepts – cohesion and coupling. Additionally, the bottleneck and cycle metrics are feasible optimisation concepts of object oriented design. The authors apply their approach within a case study to the *JHotDraw*[18] system, which comprises 207 classes and 28,776 Lines of Code (LoC). Results show that the search is able to improve the fitness function measurement in comparison to the initial fitness based on the original package

[18] http://www.jhotdraw.org/

configuration of the system. The best identified solution features a subsystem decomposition of 25 subsystems with an average subsystem size of seven classes.

Schmidt, MacDonell and Connor (2012) present a feasibility study of the application of an automatic refactoring approach to increase cohesion of packages, reduce coupling between packages and reduce the number of architecture violations in the model of a software system. The approach employed a single objective *EA* implementation. Method and constants are represented as decision variables and the genotype instance defines the assignment of methods and constants into compilation units. The approach utilises a single objective fitness function in which the three individual objectives (package cohesion, package coupling and number of architecture violations) were combined into one objective representation. The evaluation confirms the applicability of the approach to find solutions with a low number of architecture violations and acceptable cohesion and coupling based on its application in a simple model of a software system. This work informed early directions of the present research, although the focus was then redirected due to the previously discussed restrictions of single objective approaches and their unsatisfying performance that prohibited the application of the approach to non-trivial software systems.

Etemaadi, Emmerich and Chaudron (2012) and Etemaadi and Chaudron (2012) propose a conceptual framework for the application of multi-objective optimisation for the design of embedded architectures. Additionally, Etemaadi and Chaudron (2012) highlight *NSGAII* (Deb, Agrawal, Pratap, & Meyarivan, 2000) and *SPEA2* (Zitzler, Laumanns, & Thiele, 2001) as promising algorithm candidates for the implementation of such a framework. However, to this stage no further development of the approach besides the proposal of such ideas has been reported on the project webpage[19].

[19] http://www.liacs.nl/~etemaadi/home/

As highlighted above, Praditwong et al. (2011) approach software clustering from a multi-objective perspective by implementing the concepts of *Pareto Optimality* and *Non-Dominance*. Praditwong et al. (2011) highlight that in previous studies (Harman et al., 2005; Mahdavi et al., 2003; Mancoridis et al., 1999; Mancoridis, Mitchell, Rorres, Chen, & Gansner, 1998; Mitchell, 2002; Mitchell & Mancoridis, 2001a, 2001b, 2006, 2008; Mitchell, Traverso, & Mancoridis, 2001) only fixed weighted agglomerations of high cohesion and low coupling as a single objective function have been applied.

Praditwong et al. (2011) extend the *Bunch* tool by adding a multi-objective genetic algorithm implementation. The research formulates the *Equal-size Cluster Approach (ECA)* and the *Maximizing Cluster Approach (MCA)* objective settings, which can be executed with the genetic algorithm implementation. The *ECA* formulates an objective setting defined as maximizing the sum of intra-edges of all clusters, minimizing the sum of inter-edges of all clusters, maximising the number of clusters and maximising the *MQ* measure. The objectives of the *MCA* are defined as maximising the sum of intra-edges of all clusters, minimizing the sum of inter-edges of all clusters, maximising the number of clusters, maximising the *MQ* measure and minimizing the number of isolated clusters. The two multi-objective settings and the original single objective *MQ* based hill-climbing setting are evaluated based on the execution of seventeen real world problems. The systems used in the evaluation feature 20 to 198 modules and 57 to 3,262 dependencies. Each system is clustered with each of the three objective settings, a total of thirty times. The results are compared on the basis of the average *MQ* performance and the calculation of statistical significance by applying a *t*-test. The results indicate that the *Equal-size* objective setting produces a better final solution, based on the best found *MQ* value, than the existing single-objective hill-climbing approach and the *MCA* implementation. An evaluation of the spread or convergence of the *Pareto-Front* is not conducted as only the final best solution based on the *MQ* value is considered for the evaluation of the three objective settings.

However, the consideration of the complete *Non-Dominated Pareto-Front (NDPF)* is more appropriate to make conclusions on the performance of a *MOEA* implementation. Additionally, the inclusion of the development of the search instead of only considering the final search outcome is also a valuable aspect to assess the performance of a *MOEA* configuration. Hence, consideration of the performance of the complete optimal *Pareto-Front* as well as the speed of convergence are valuable areas of research to extend the understanding of the general performance of *MOEA* implementations.

Barros (2012) extends the work of Praditwong et al. (2011) and compares the performance of a multi-objective clustering approach with three different objective configurations. The first objective configuration features four objectives: coupling (to be minimised), cohesion (to be maximized), cluster size difference (to be minimised), and number of clusters (to be maximized). In the second configuration the *MQ* metric was added as a fifth objective (to be maximized). Similarly, in the third configuration the *EVM* metric was added to the original first objective configuration as a fifth objective (to be maximized). Barros (2012) employs the *NSGAII* algorithm implementation from the *JMetal* framework to evaluate the three different objective configurations by applying each of the objective configurations to a set of 14 software system instances. These software system instances feature 26 to 195 classes and 61 to 1,137 type dependencies. Each pair of an objective configuration and a software system instance is executed 30 times, due to the probability-based concepts of the implemented *NSGAII* algorithm, to enable general statements on the performance of the individual objective configurations to be made. An optimal *Pareto-Front* is built for each of the executed objective configuration and system instance pairs by joining the fronts generated by each executed run of the corresponding configuration. Finally, Barros (2012) removes the *MQ* and *EVM* objective settings from the corresponding fronts and the three fronts are merged to generate a final best optimal *Pareto-Front*. The merged best optimal *Pareto-Front* is compared with the optimal *Pareto-Fronts* of the three configurations in terms of both the *Error Ratio* and *Generational Distance.* Furthermore, the Mann-

Whitney p-value and the effect size for the *Error Ratio* and *Generational Distance* are calculated by applying a pairwise comparison of the three objective configurations across all 14 software system instances.

Based on the presented data, Barros (2012) concludes that suppressing *MQ* and *EVM* from the search process can help to find solutions with improved *Error Ratio and Generational Distance*. Nevertheless, it needs to be raised that objective configurations that include the *MQ* and *EVM* metric might have had a disadvantage as the additional objective adds a dimension of complexity. Hence, the research design utilised by Barros (2012) might have disadvantaged the objective configurations that include the *MQ* and *EVM* metric. Additionally, it needs to be raised that the application of the *Error Ratio* and *Generational Distance* features some limitations. Section 3.3.4 gives a more detailed and graphical explanation of such limitations. Therefore, the application of performance metrics (e.g. Additive Epsilon Indicator and/or *Hypervolume*) that evaluate the relative convergence of *Pareto-Fronts* might give more insight into the comparative performance of different search configurations.

Despite these limitations, the objective stated in Barros (2012) has relevance for the present research. Barros (2012) conducted this research to evaluate if objective configurations that focus less on reducing dependencies between modules are able to deliver better module configurations. The present research adopts a similar stance, noting that a good architecture design or even a good modularisation should not focus exclusively on the minimisation of dependencies between modules. Instead tolerable dependencies between modules may well exist if they align with the desired system design, while other dependencies may be less favourable e.g. cyclic dependencies or dependencies that violate the conceptual architectural design.

In general, interesting techniques and approaches have been proposed to overcome challenges in the application domain of search based software modularisation. Nevertheless, it is evident that these approaches do not permit any comparison of

solution outcomes across the individual studies. Hence, a framework that allows the performance evaluation of SBSE approaches would be helpful to assess and compare the feasibility of the developed approaches.

## 2.5 *Summary of Related Work*

This chapter reviewed literature in the areas of software erosion, design principles of software architectures, principles of *SBSE* and *MOEAs*. The review of the literature helped to identify research gaps that are central to the formulation of the research objective of this research.

In summary, *SBSE* driven modularisation approaches feature pre-defined sets of objectives. The only parameters to influence the search in the reviewed single objective studies is the definition of weights for the components of the single objective function, (Seng et al., 2005), or the definition of the strength of dependencies between low-level artefacts (Mitchell, 2002). Evident in most studies that applied optimisation strategies in the area of software clustering is the creation of solutions based on the rewarding of high cohesion and low coupling of interrelated implementation artefacts. Harman, McMinn, de Souza & Yoo (2012) and Abdeen et al. (2009) state that the designed fitness functions are in fact not applicable to capture a complete representation of good software design principles and that more flexible approaches are required to identify promising solutions. In terms of the objective of this research these previously applied metrics can conflict with the general design principles of high-level architecture models. Reasons for this are, for example, no consideration of the natural dependency flow from higher implementation artefacts to lower implementation artefacts, no consideration of allowed and forbidden dependencies between high-level architecture artefacts, and no consideration of the structure within high-level artefacts. Hence, the extension of fitness functions to incorporate metrics that more effectively consider concepts associated with high-level architectural design and that battle indicators of architectural erosion, as presented in section 2.3, is worthwhile.

The related research that has been conducted in the domain of *SBSE* and software partitioning (single- and multi-objective) focusses on the composition of low-level artefacts (compilations units) into the next higher abstraction level (packages or folders). However, software architectures feature abstractions on multiple levels (e.g. packages, subsystems, layers, layer groups). An approach commonly adopted in software architecture design is to group packages or folders into conceptual architecture models, which feature a defined set of subsystems and dependencies between these high-level artefacts. These conceptual architecture models are a representation of the desired design of the system. The design of this conceptual architecture model is driven by desired domain and distribution aspects of the system.

The reviewed approaches discover a model of the next higher level of abstraction as part of the solution. The extension of previous work by enabling the reconstruction of multiple models of different abstractions and the classification of software artefacts into these models is worthwhile. The consideration of predefined conceptual architecture models is also a more realistic adaption of the manual software architecture reconstruction process in which low-level artefacts are manually classified into artefacts at higher abstraction levels.

The design of software architecture configurations depends on a variety of potentially conflicting requirements. Within this research it is anticipated that a predefined configuration of objectives and objective weights has limited validity to satisfy the requirements of development stakeholders in every software modularisation or architecture reconstruction scenario. Furthermore, it is hypothesized that the quality of an architecture design cannot be expressed in one single fitness value. It is therefore suggested in this research that the application of *MOEAs* in combination with the application of established high-level software architecture design metrics is a promising area of research.

Barros (2012) applied a *NSGAII* algorithm and Praditwong et al. (2011) applied a TWO Archive based *GA* implementation in the problem domain of software modularisation. However, a wide range of other *MOEA* implementations exists, including *Differential Evolution*, *Scatter Search*, *Particle Swarm Optimisation* and *Decomposition* based *EAs*. Nothing is currently known regarding the performance of such *MOEA* implementations in the targeted application domain, and more generally very little is known in relation to the performance of *MOEA* implementations when applied to other software engineering problems. Hence, it is worthwhile to thoroughly evaluate and compare the performance of other *MOEA* implementations and tunings in the targeted application domain. This might also lead to insights that enable conclusions to be drawn regarding the general performance of *MOEA* implementations in other related application domains.

Additionally, a variety of multi-objective approaches exist that might be leveraged to address problems in the domain of software engineering. However, different experimental designs are employed to demonstrate the feasibility of the developed approaches. For example, Barros (2012) applies *Error Ratio* and *Generational Distance* to evaluate the optimal *Pareto-Fronts* that are achieved in the two applied fitness function settings. On the other hand, Praditwong et al. (2011) rely on the *MQ* metric to assess the performance of their approach. It is anticipated that a comprehensive and flexible multi-objective evaluation framework that would enable the comparison of solution sets independently of the solution implementation would be a valuable contribution to the *SBSE* community. It would be valuable to include additional multi-objective performance metrics, e.g. *Inverted Generational Distance*, *Additive Epsilon Indicator*, *Spacing* and *Hypervolume* (compare: section 3.3.4), in such a framework.

The present research also assumes that a variety of feasible solutions are discovered in the search. Such solutions might or might not necessarily be included in the optimal *Pareto-Front*. Hence, it is worthwhile to evaluate the feasibility to implement an approach that enables development stakeholders to filter and review the set of

discovered solutions efficiently based on solution attributes that extend the defined objective configuration.

A variety of different search based driven partitioning, modularisation and decomposition approaches have been reviewed in this chapter. At this stage none of the described approaches is available to the public. One of the outcomes of this research is a candidate prototype, namely the *Rearchitecturer* system, that addresses and evaluates the points of contribution addressed above. The availability of such a candidate prototype as an extendable open-source artefact is valuable in enabling replication and extension of the research conducted and reported here as well as supporting comparisons with other research efforts.

# 3 Methodology

An appropriate research methodology needs to be implemented to demonstrate and evaluate the relevance and effectiveness of the approach developed in this research in a robust and rigorous way.

The main objective of this research is to evaluate the feasibility of multi-objective optimisation strategies when applied in the area of architecture reconstruction to identify feasible architecture classifications that can operate as a starting point for the modularisation of software systems and the containment of software erosion. Through the course of this research a software prototype is implemented that features architecture reconstruction functionality based on the application of multi-objective optimisation methods. This prototype is the basis for the evaluation of the objective of this research. Hence, this research is both exploratory and constructivist in nature. Therefore, the objective of the present research fits the frameworks and guidelines of design science in the information systems discipline (Hevner, March, Park, & Ram, 2004; Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007; Vaishnavi & Kuechler, 2008). These research guidelines are used extensively in high quality research across the domains of software engineering and information systems development and management (Antunes, Zurita, & Baloian, 2014; Melville, 2010; Pruijt & Brinkkemper, 2014).

The present research therefore applies the *Design Science Research Methodology (DSRM)* as specified by Peffers et al. (2007), consisting of six stages that operate as a template for the conduct, presentation and evaluation of design science research in the information systems discipline. Figure 17 depicts these six stages and the nominal process of design science research as presented in Peffers et al. (2007):

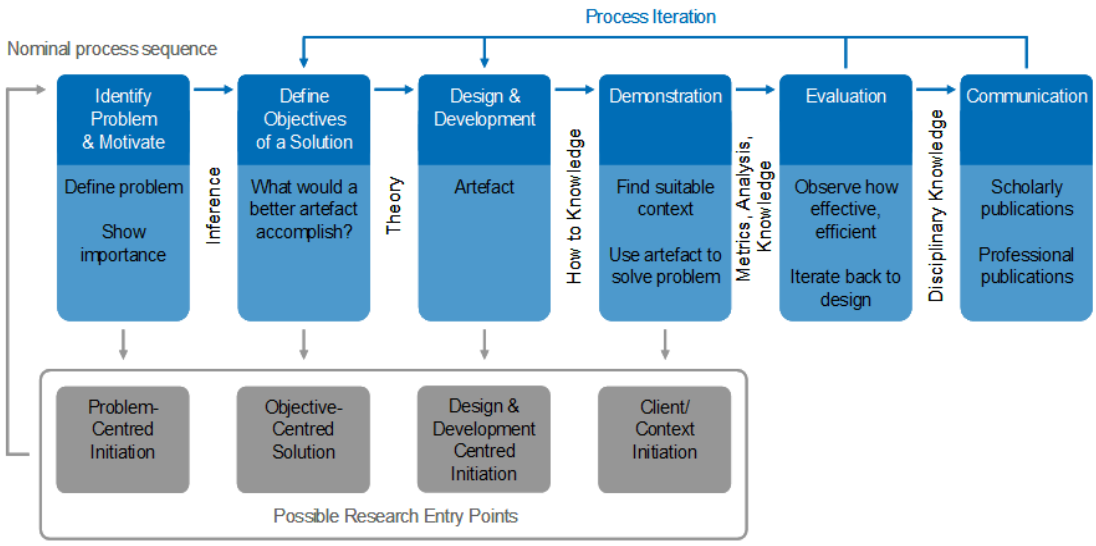


**Figure 16: Research process of the DSRM (Peffers et al., 2007)**

The first step of the research process described in Peffers et al. (2007) serves to identify and motivate the relevance of the addressed research problem. The five subsequent steps (Define Objective of a Solution, Design & Development, Demonstration, Evaluation, and Communication) are conducted in an iterative manner. Each phase is revisited a number of times and refined throughout the duration of the research as new understandings are discovered during the review and analysis of the outputs of the individual phases. This iterative approach enables the adaptation and refinement of the design of the artefact under development.

The individual phases need to be populated with problem context specific activities to enable the conduct of the research. The following sections describe these context specific activities that have been employed in the present research. The sequence of the presentation adapts the nominal sequence of the design science research process as described by Peffers et al. (2007). The identification of a problem and the definition of an objective as well as the design, development and demonstration phase are highly interconnected in the present research and are therefore presented conjointly in the following sections. Additionally, a contribution of this research is a multi-objective evaluation framework that enables the flexible and repeatable analysis and comparison

of result datasets. Hence, the methods and techniques that are utilised to evaluate the designed architecture reconstruction artefact and to collect datasets are of particular relevance for the implementation of this research.

## 3.1 *Identification of the Problem and Definition of the Research Objectives*

Relevant literature needs to be reviewed to identify a research problem in the targeted problem domain and to define the research objectives. Hence, this research project started with an in-depth literature review to ensure familiarity with the philosophies that underpin this domain, existing research, challenges, opportunities and open questions. This project draws on a number of different research areas within the software engineering discipline, particularly software architecture management and monitoring, architecture reconstruction, software modularisation and partitioning, software quality assessment, search based software engineering and, most importantly, research that combines these fields. The reviewed literature also enables justified claims to be made regarding the novel contributions of the solution presented in this research. The relevant findings of the reviewed literature are presented in Chapter 2 of the thesis. Based on the reviewed literature, research gaps have been identified that are central to the formulation of the research objective. The summary of the identified research gaps and the subsequent formulation of the objectives of the present research are presented in section 2.5 of this thesis.

## 3.2 *Design, Development and Demonstration*

A prototype, namely the *Rearchitecturer* system, is designed and developed to enable the evaluation of the objectives of the present research. The developed prototype enables the reconstruction of software architecture configurations based on the application of multi-objective optimisation techniques. Different compositions of technologies and components are feasible in supporting the development of such a candidate prototype. However, the designed prototype can only represent one feasible

composition of such technologies. Hence, the selection of state-of-the art-technologies is crucial if the research is to produce a high performing instance of the developed theoretical framework. Existing technologies and methods are reviewed and evaluated within this research stage in order to identify an efficient and useful orchestration of technologies and to reuse existing components for the development of candidate prototypes. Various concepts that are essential for the implementation of a search based driven architecture reconstruction approach, such as high-level architecture patterns, architecture design metrics, methods to define conceptual architecture models, approaches to conduct compliance checking of conceptual and physical architecture models, and *MOEA* approaches and principles, are all reviewed and documented as part of the literature review presented in Chapter 2. A variety of open-source libraries exist that facilitate such functionality. Additionally, a variety of programming paradigms and languages exist in the software engineering discipline. Correspondingly, different architecture designs and patterns are favoured depending on the applied programming paradigm and system distribution. The scope of this research is not to deliver a prototype that enables a solution for every one of these programming languages, programming paradigms, architecture styles and patterns. The objective of this research is to evaluate the feasibility of a search based driven architecture reconstruction framework. For the evaluation of this research a representative programming language and paradigm combined with a selection of architecture styles and architecture patterns is sufficient to demonstrate the designed theoretical framework. The evaluation of this work focuses exclusively on software systems that are developed in *Java.* A variety of open-source systems exist that have been developed in *Java*. Many of these systems have been actively developed for several years and most likely face the known problems of software erosion. Such open-source systems represent a viable pool of candidate systems for the evaluation of the developed prototype.

Chapter 4 discusses aspects of the design and implementation of the *Rearchitecturer* prototype in detail. Additionally, Chapter 4 demonstrates the functionality of the *Rearchitecturer* component and highlights how the functionality and evaluation aspects contribute to the domain of multi-objective software modularisation and architecture reconstruction.

## 3.3 *Evaluation of Prototype*

The objective of the system analysis and evaluation stage is to gather data that enables the formulation of conclusions on the feasibility, contributions and limitations of the developed approach. This data is collected through the application of the developed candidate prototype. The evaluation of the prototype reveals the capability of the constructed theoretical framework to overcome the problems addressed by the research objective. The definition of suitable methods of system analysis and evaluation is crucial to enable a thorough and sound data collection, analysis and conclusion on the feasibility of the developed approach.

The *Rearchitecturer* component features a range of software architecture metrics, reconstruction strategies, *MOEA* implementations and variation operations. The flexible design of the component enables the definition of a variety of optimisation configurations. The execution and analysis of all combinations of these feasible setups exceeds the scope of the present research. Hence, it is necessary to identify evaluation scenarios that contribute data towards the evaluation of the formulated objective. Furthermore, methods need to be identified to conduct the analysis of the collected data to enable conclusions regarding the feasibility of the designed approach. The selection of suitable application scenarios, *MOEA* implementations, *MOEA* parameter tunings, evaluation systems, but also the evaluation and analysis techniques themselves, is discussed in the following sections.

### *3.3.1 Architecture Reconstruction Application Scenarios*

The present research applies the concepts presented in Harman and Clark (2004) that outline the application of software metrics as fitness functions to determine the fitness of a generated solution. Software design metrics that asses the quality of software architecture compositions are relevant in the evaluation. An overview of established software architecture metric implementations has been given in section 2.3. These seven architecture design metrics are employed as optimisation goals in the evaluation of this research.

The approach developed in this research facilitates the classification of software artefacts into conceptual architecture models on multiple abstraction levels. Additionally, conceptual architecture models can be reconstructed as part of the reconstruction process. However, following Breivold, Crnkovic, and Larsson (2012), Fowler (2002) and Martin (2011), this research perceives the conceptual architecture model as a blueprint of the desired design of the system. A conceptual architecture model is supposed to describe the design of the system based on domain aspects (Martin, 2011). Hence, the design of the conceptual model is ideally driven by domain aspects and their relationships. The implementation of the system should adapt to this blueprint. Correspondingly, the conceptual model should not reflect the implementation of the system. Hence, from an architecture design perspective the rebuilding of the conceptual architecture based on the structures of the physical source code artefact conflicts with the ideas presented in the mentioned architecture design literature. Therefore, experiments are conducted in the evaluation of this research that consider predefined conceptual architecture models. These conceptual architecture models operate as target architectures during the classification process.

The conceptual architecture model can be utilised to model different system architectures. For example, a transient architecture style can be utilised if the application is running on one machine and no machine boundaries exist (Fowler, 2002). A strict layer dependency, in which a layer can only access artefacts in the directly

depending layer, can be applied to model machine boundaries (Fowler, 2002). The structure of the conceptual architecture is considered by some of the employed architecture design metrics. Hence, the employment of different architecture styles within the evaluation is useful to reveal information on the applicability of the approach within different architecture styles.

Principles of good architecture design envisage that classification of physical source code artefacts into conceptual high-level artefacts is driven by the intended functionality of the physical artefacts (Martin, 2011). The optimisation implemented in this research is based on established architectural design indicators and does not automatically consider functionality aspects of the physical artefacts. However, stakeholders might have an understanding of this intended functionality for some of the physical artefacts that they want to have included in the solution. Hence, stakeholders can assign artefacts prior to the execution of the search to include their domain knowledge into the architecture reconstruction process. The developed approach supports such predefined assignments. As a result any visited solution will feature the predefined artefact assignments. The consideration of such a manual assignment impacts the solution space of the search and impacts the performance of the employed search. Hence, the inclusion of an experiment scenario that considers predefined artefact assignments is helpful to gather data on the performance of the employed *MOEA* implementations.

The analysis of the actual functionality of the physical artefacts of the analysed software systems, definition of a corresponding conceptual architecture and assignment of a subset of the physical artefacts into suitable subsystems is beyond the scope of the present research. However, a randomised assignment of physical source code artefacts combined with the subsequent execution of the optimisation is a useful method to generate data that enables conclusions to be drawn on the performance of *MOEA* implementation operating in constraint classification scenarios. It is necessary to conduct multiple iterations of the randomised assignment of physical source code

artefacts and the subsequent optimisation to gain a set of results that enables representative conclusions.

### *3.3.2 Applied MOEA Implementations*

The previous sections identified application scenarios to evaluate the feasibility of the designed approach in an architecture reconstruction problem context. *MOEA* implementations need to be employed to run the optimisation in the outlined architecture reconstruction scenarios.

The application of optimisation approaches is appropriate where there is no known deterministic method that will generate a good solution in an acceptable amount of time and where the complexity of the problem is too high to apply a brute force approach. These problem scenarios leave optimisation approaches as the last resort to avoid the application of a solution generation approach that relies explicitly on guessing (random search) (Luke, 2010). Arcuri and Briand (2011) describe random search implementations as an essential benchmark to determine the relative usefulness of randomised optimisation implementations. Additionally, however, Arcuri and Briand (2011) highlight that comparative studies of selected optimisation implementations with a random search implementation are rarely conducted.

The optimisation community has developed a variety of different *MOEA* implementations in recent decades. As presented in section 2.4.2, Sayyad and Ammar (2013) reviewed 51 research papers that apply multiple optimisation techniques in the *SBSE* domain and report that a total of 25 different *MOEA* implementations have been applied. The most commonly applied *MOEA* implementations are NSGAII (34), SPEA2 (9), MOCELL (7), PAES (4) and MOGA (4). The other twenty *MOEA* implementations are only utilised once or twice in the reviewed studies. Sayyad and Ammar (2013) also report that only 30% of the research efforts in the multiple objective *SBSE* discipline apply more than one *MOEA* implementation.

Two multi-objective *GA* implementations have been applied in the related problem domain of software modularisation. Barros (2012) applied NSGAII and Praditwong et al. (2011) applied a TWO Archive based *GA* implementation in the problem domain of software modularisation. However, there are no comparisons of different *MOEA* implementations in any known research (Sayyad & Ammar, 2013).

The absence of comparative studies that analyse the performance of multiple *MOEA* implementations encourages the employment of different types of *MOEA* implementations in the targeted problem domain. However, the lack of consistent evaluation approaches to evaluate different *MOEA* implementations or settings makes it difficult to compare results and make general statements regarding the general feasibility of approaches. Hence, a novel multi-objective evaluation framework is a further valuable contribution to this research area.

The approach developed in this research integrates two optimisation frameworks (*MOEA* and *JMetal*) (compare: section 4.1.4). One advantage of the utilisation of these frameworks is that a variety of established *MOEA* implementations is available to be applied in the addressed problem domain. Nevertheless, the employment of each of the many *MOEA* implementations available is beyond the scope of this work.

The objective configuration that is applied in the evaluation of this research features a total of eight objectives. Basic *MOEA* implementations have difficulties in achieving good convergence in complex optimisation configuration settings (Hughes, 2005; Li & Zhang, 2009). He and Yen (2014) quote relaxed concepts of *Pareto-Dominance*, decomposition based concepts, performance indicator based concepts or grid based concepts as potentially helpful to overcome the challenges of many objective configuration settings. In the evaluation of this research a mixture of established *MOEA* implementations (*NSGAII - Genetic Algorithm*, *AbYSS – Scatter Search*, *GDE3 - Differential Evolution*) in combination with *MOEA* implementations that apply decomposition based concepts *(MOEAD)* and ε-dominance (*OMOPSO - Particle Swarm*

*Optimisation*) as a relaxed form of *Pareto-Dominance* are applied. Additionally, *RandomSearch* is applied as a benchmark implementation. Thus, in total a set of six *MOEA* implementations are employed in the evaluation of this research that feature a variety of different optimisation strategies.

### *3.3.3 Evaluation Systems*

The defined problem scenarios and MOEA configurations are applied on a set of software systems to enable conclusions to be drawn on the performance, applicability and scalability of the developed approach.

It has been found that the consideration of the systems or metrics incorporated in existing benchmark suites such as the Qualitas Corpus or the Dacapo benchmark is not feasible. The Qualitas Corpus benchmark suite provides size metrics of multiple system releases. In the present research the reconstruction of only one recent release is examined, as the objective of this research does not focus on the transformability of software systems at different stages of the system lifecycle. In addition, the Qualitas Corpus suite features a total of 112 examined systems. The examination of such a high number of systems is computationally infeasible as the evaluation design requires the execution of all experiment settings for each system. The Dacapo benchmark suite provides a set of runtime metrics. The approach considered in the present research focuses on the optimisation of static high-level quality indicators. While the assessment of dynamic metric benchmarks and comparison with convergence of static metrics is worthwhile it is argued in this research that the consideration of dynamic quality benchmarks would add limited insights to demonstrate the feasibility of a multi-objective architecture reconstruction approach. Hence, it has been concluded in this research that the application of the approach to benchmark suites would not add value to the evaluation of this research. Instead, recent releases of open-source systems have been utilised as evaluation systems. The evaluation of the systems has been started on smaller systems to prove the general feasibility of the approach. The size of the evaluation systems has been increased incrementally to demonstrate the scalability of

the developed approach. As a result, the approach has been evaluated on the following five software systems: Apache Log4j, Apache Commons Math, Apache Ant, Lucene and Rearchitecturer.

All but the *Rearchitecturer* system are established open-source projects with an active user community. Multiple developers are permanently involved in the maintenance and enhancement of these projects. These systems each feature a module-based architecture that is publicly available on the corresponding system webpages. The modules in these systems are maintained as separate projects. The dependencies between the modules are organised with *Maven*[20]. Hence, at a project level the systems are cycle free and have a defined dependency structure. However, none of the projects employs a dependency management tool (such as *Sonar*, *XRadar* or *DependoMeter)*. Hence, the smallest unit from a high-level architecture perspective is a *Maven* project. Dependencies between packages of the same module are not restricted and monitored. It is unknown if metrics are applied to review the architectural quality of the individual modules in these systems.

The *Rearchitecturer* system, on the other hand, has been developed as a prototype for the evaluation of this research. Thus, one developer was involved in the development of the *Rearchitecturer* system and no active user-community exists at this stage. In summary, the selected systems are of different size, structure and maturity. Table 1 depicts software metrics that describe aspects of the size and structure of the selected software systems.

[20] http://maven.apache.org

**Table 1: Size Metrics of Evaluation Systems**

| Name | Number of Packages | Lines Of Code | Number of Types |
|---|---|---|---|
| **Apache Ant v.1.9.2** | 276 | 131,212 | 1,772 |
| **Apache Math v.3.2** | 140 | 171,171 | 2,106 |
| **Apache Log4j v.1.2.17** | 40 | 30,456 | 453 |
| **Lucene v.4.4.0** | 50 | 151,340 | 2,264 |
| ***Rearchitecturer*** | 51 | 33,231 | 537 |

### *3.3.4 Optimisation Performance Metrics*

The empirical performance evaluation of single objective optimisation focuses on the quality assessment of the final solution based on the achievement within the desired objective and the computational effort expended to achieve solutions (Coello et al., 2007; Talbi, 2009).

Consideration of absolute achievement in a desired objective has the advantage that the achieved values are specific to the problem and hence allow an assessment of a solution from a domain perspective. For example, stakeholders might not be satisfied with a minimum number of 500 architecture violations in an optimisation run that features the minimisation of architecture violations as the only optimisation goal. It seems likely that only solutions that feature close to zero architecture violations would be considered as a useful outcome from a software engineering perspective. Hence, the consideration of absolute achievement in each of the individual objective dimensions, based on the presentation of descriptive statistics such as minimum, maximum, mean, median and standard deviation of the populations, is an important instrument to reveal information on the feasibility of the approach.

In addition, and a key differentiator from prior research, the present research applies multi-objective optimisation techniques. The assessment of a multi-objective solution set based on the assessment of achievement in the individual objectives has

limited value in terms of making conclusions on the overall performance of a *MOEA* optimisation setting. The review of a dataset from the individual objectives perspective does not permit any conclusions to be drawn on the performance of the promising solutions of this dataset in other objective dimensions.

To thoroughly evaluate the performance of a multi-objective approach, all of the employed objectives need to be taken into consideration. Hence, the performance assessment of multi-objective based optimisation approaches is more complex. The major difficulty of multi-objective assessment is that the output of the optimisation process is not a single solution but rather a non-dominated *Pareto-Front* (compare section 2.4.2). Multiple non-dominated *Pareto-Fronts* need to be compared to evaluate the performance of different multi-objective metaheuristics. Figure 17 shows two non-dominated *Pareto-Fronts* of a two objective minimisation problem. None of the presented *Pareto-Fronts* dominates the other completely. This makes a comparison of the two approximation sets challenging (see: Figure 17).

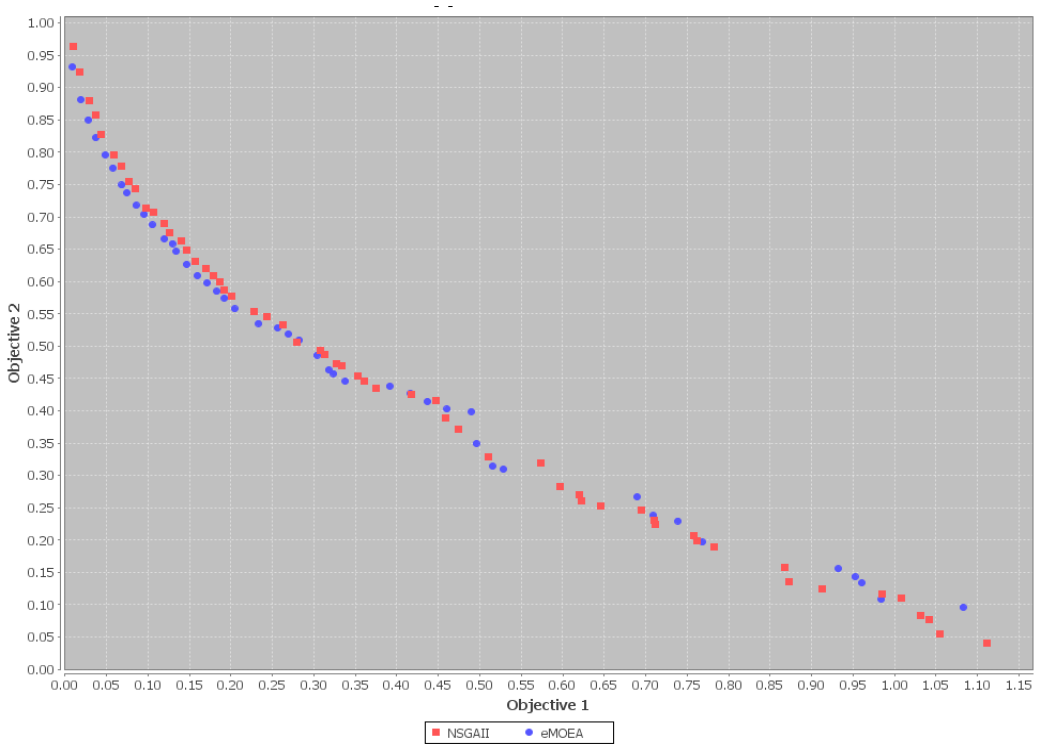


**Figure 17: Non-dominated *Pareto-Front* of two different algorithms**

Performance indicators offer support when assessing the quality of *Pareto-Fronts* (Coello et al., 2007; Okabe, Jin, & Sendhoff, 2003; Talbi, 2009). Generally, performance indicators that assess convergence and diversity aspects of *Pareto-Front*s can be differentiated. However, hybrid forms, which express convergence and diversity aspects in one metric, also exist (Luke, 2010). Additionally, performance indicators that require or do not require the true *Pareto-Front* to be calculated can be differentiated. The true *Pareto-Front* is understood as the best achievable *Pareto-Front* of a problem (Coello et al., 2007). Often the true *Pareto-Front* is not known or cannot be calculated for a problem. This certainly applies in the targeted problem domain, as the true *Pareto-Front,* that entails all non-dominated architecture configuration solutions for a selected optimisation configuration, is not known and cannot be generated deterministically. The developed evaluation approach creates a super-*NDPF* of combined *NDPFs* from the individual optimisation runs that can be considered an approximation of the true *Pareto-Front*. The creation of such a super-*NDPF* is an established technique of evaluation and has been applied in other optimisation research (Barros, 2012; Ferrer, Chicano, & Alba, 2012; Sayyad, Menzies, & Ammar, 2013). A limitation of utilising an approximation instead of the true *Pareto-Front* is that the performance indicator only calculates a relative convergence and diversity. Hence, results that have been created with different super-*NDPFs* are not directly comparable. In the present research this needs to be considered if *NDPFs* are calculated with different objective settings or with different software systems. Hence, an approach is suggested that relies on the application of normalisation techniques to enable the comparison of performance indicator results from such incompatible *NDPF* calculations.

This research evaluates the feasibility of a multi-objective reconstruction approach based on the utilisation of architecture design metrics as optimisation goals. In the evaluation, established performance metrics are considered to enable statements on the performance of the applied algorithms. The performance metric implementations of

the *JMetal*[21] and *MOEA* framework are utilised in the present research. As noted above, Sayyad and Ammar (2013) recently reviewed 51 research papers that applied multi-objective optimisation techniques and analysed, among other things, the method of evaluation evident in these papers. Sayyad and Ammar (2013) state that 15 of the 51 research papers applied multi-objective performance metrics to conduct the evaluation. Different performance metrics, such as *Spacing*, *Error Ratio, Generational Distance*, *Inverted Generational Distance* and *Hypervolume* are applied in these studies.

The following sections give an overview of some of the more commonly used performance metrics that are applicable to evaluate the feasibility of the developed approach.

### *Spacing*

The *Spacing* metric describes the spread of the solutions within the objective space of a non-dominated approximation set (Zitzler, Thiele, Laumanns, Fonseca, & Da Fonseca, 2003). A higher value indicates better spread in the objective space, whereas a low spacing value indicates that all solutions of the analysed approximation set are close together and feature a small spread in the objective space. Hence, the comparison of the spacing metric for a set of algorithms when applied to the same search problem can indicate whether an algorithm can find solutions that feature a better spread along the objective space. However, it is also suggested in the present research that more condensed *NDPF* are potentially more desired in a multi-objective software modularisation problem as stakeholders prefer modularisation solutions that feature a well-balanced achievement in the defined objectives. Section 4.2.2 discusses this problem characteristic in more detail.

Additionally, if all algorithms under consideration feature a low spread it does not necessarily indicate poor performance of the algorithms themselves, but rather denotes

[21] http://jmetal.sourceforge.net/

that no great changes in the objective space could be achieved. Hence, the spacing metric is also a measure to characterise the complexity of the search landscape.

### *Contribution and Error Ratio*

The *Contribution* metric is a convergence metric that reports the ratio of the number of solutions of an optimal *Pareto-Front* of the algorithm under consideration that are also members of the true *Pareto-Front* (Van Veldhuizen & Lamont, 1999). On the other hand, the *Error Ratio* reports the ratio of members that are not in the true optimal *Pareto-Front* and the total number of solutions of the optimal *Pareto-Front* of the algorithms under consideration. Nevertheless, both metrics express exact matches in the optimal *Pareto-Front* calculated by the algorithm under consideration and a best known *Pareto-Front*.

A general critique of the *Contribution* and *Error Ratio* performance metrics is that they report a binary match of solutions in *Pareto-Fronts. S*olutions that are *close to* the best known optimal *Pareto-Front* but *not included in* the best known *Pareto-Front* are not considered (Van Veldhuizen & Lamont, 2000). Hence, the *Contribution* and *Error Ratio* performance indicators do not consider relative convergence and have limitations due to their reliance on reporting binary convergence. Barros (2012) constitutes the only related research that applied the *Error Ratio* performance metric in an evaluation.

### *Generational Distance (GD)*

The *Generational Distance (GD)* calculates the average population distance from the optimal *Pareto-Front* calculated by an algorithm under consideration to a best known *Pareto-Front* (Van Veldhuizen & Lamont, 2000). The best known *Pareto-Front* can be the true *Pareto-Front* or an approximation based on the calculation of multiple algorithms. The sum of the minimum distance is calculated from every member of the optimal *Pareto-Front* to the closest member in the best known *Pareto-Front*. Figure 18 illustrates the calculation of distance for the *GD* metric based on a two-objective minimisation

problem. The calculated sum is divided by the number of members of the optimal *Pareto-Front*.

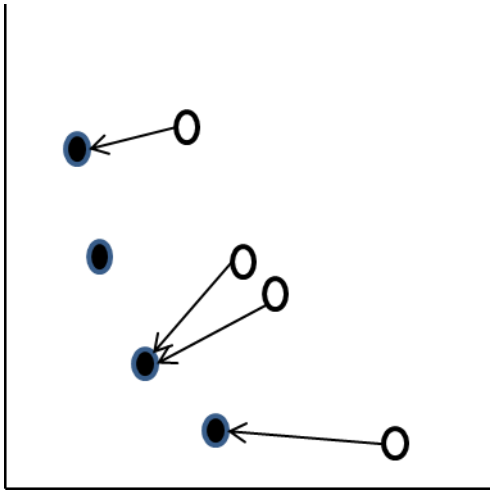

**Figure 18: Calculation of *Generational Distance* metric**

One point of critique in the rigour of the *GD* performance metric is that it does not consider the spread, shape and number of solutions in the best known *Pareto-Front* (Zitzler, Brockhoff, & Thiele, 2007). Hence, the *GD* metric reports good performance if the members of the optimal *Pareto-Front* under consideration are close to any solution of the best known *Pareto-Front*. This is particularly relevant if the optimal *Pareto-Front* does not feature a wide spread or only features a small number of solutions. Hence, the similarity of the spread, shape and the number of solutions of the two *Pareto-Fronts* is not considered. An explicit reliance on the *GD* performance metric is not recommended.

Sayyad et al. (2013) reports that the *GD* performance metric is applied in a total of four *SBSE* papers in the problem domains of testing, software project management and software clustering. Barros (2012) applies the *GD* performance metric in the related problem domain of software clustering.

### *Inverted Generational Distance (IGD)*

The *Inverted Generational Distance (IGD)* calculates the distance from the best known *Pareto-Front* to the optimal *Pareto-Front* (Li & Zhang, 2009). This addresses the shortcoming of the *GD* metric mentioned in the previous section if the best known *Pareto-Front* features a different shape or wider spread along the objective space than the optimal *Pareto-Front*. Hence, it is relevant to consider the *IGD* metric during the evaluation process. If the result is similar to the *GD* metric it can be concluded that the shape and spread of the best *Pareto-Front* and the known *Pareto-Front* are commensurate. A noticeable disagreement in the *GD* and the *IGD* measurement indicates that the shape and spread of the best known *Pareto-Front* and calculated *Pareto-Front* are different, which is an unfavourable performance characteristic of the evaluated algorithm. Sayyad and Ammar (2013) report the application of the *IGD* metric in the problem domains of testing (Assunção, Colanzi, Pozo, & Vergilio, 2011), software deployment and configuration in the cloud (Frey, Fittkau, & Hasselbring, 2013).

### *Epsilon Indicator*

The *Epsilon Indicator (EI)* calculates the minimum factor by which an approximation set has to be translated into the objective space to dominate another currently dominating approximation set (Coello et al., 2007). The *EI* metric is motivated by the assumption that an approximation, that dominates another, is more favourable and features absolute better solutions. In the scenario of a performance evaluation of an algorithm, the *EI* metric is calculated for an optimal *Pareto-Front* that is known and the true *Pareto-Front*. The true *Pareto-Front* can be a consolidation of the optimal *Pareto-Front* from various algorithms or the absolute true *Pareto-Front*. The *EI* is calculated based on the worst case solution distance of the multi-objective space to make the approximation set dominate the best-known approximation set. Hence, a large gap between two approximation sets leads to a poor *EI* metric value. Figure 19 illustrates the calculation of the *EI* based on a two-objective minimisation problem.

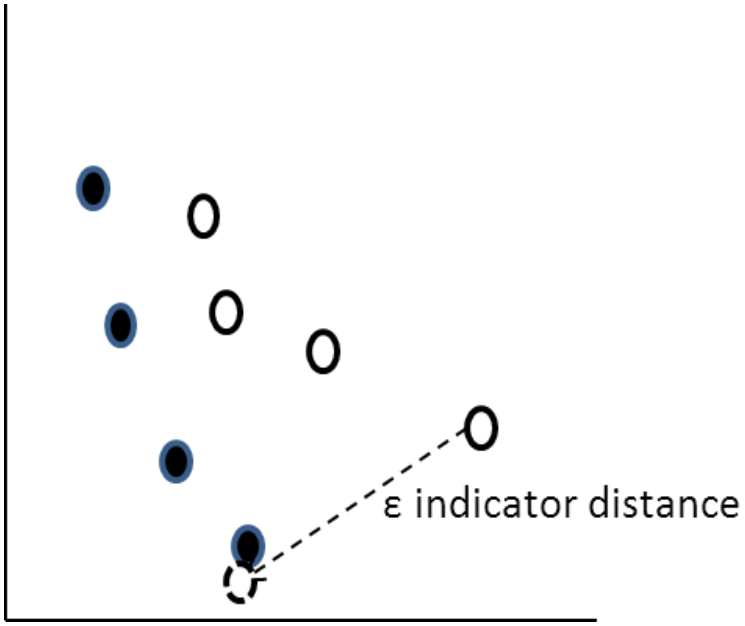


**Figure 19: Calculation of the *Epsilon Indicator* metric**

Based on the review conducted by Sayyad and Ammar (2013) the *EI* performance metric is applied as a performance indicator in *SBSE* related research.

### *Hypervolume*

*Hypervolume* is defined as the n-dimensional space that is confined by an n-dimensional set of points (Zitzler, Brockhoff, & Thiele, 2007). When applied to multi-objective optimisation, the n-dimensional objective values of the individual solutions of the *Pareto-Front* solutions are understood as vectors. The space that spans along these vectors in relation to a reference point, usually the anti-optimal point or worst possible point for the space, is the *Hypervolume*. Hence, the *Hypervolume* calculates the space that is weakly dominated by a *Pareto-Front*. The *Hypervolume* is reported as a normalised value based on an approximated optimal *Pareto-Front* or the true *Pareto-Front*. Correspondingly, the *Hypervolume* metric expresses in a single unary value a measure of the spread of the solutions along the *Pareto-Front*, as well as the distance of a *Pareto-Front* from an optimal *Pareto-Front*. Hence, the *Hypervolume* is maximised only if the set of solutions contains all Pareto optimal points. Figure 20 pictures the calculation of the *Hypervolume* metric based on two *Pareto-Front* solution sets.

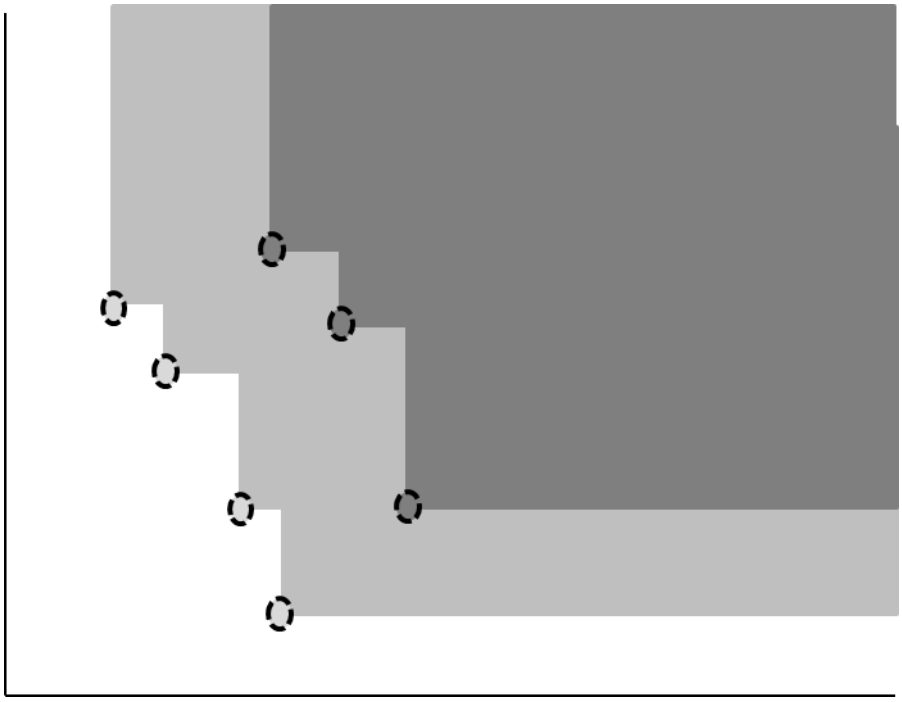

**Figure 20: Calculation of the *Hypervolume* metric**

Zitzler et al. (2007) argue that the *Hypervolume* metric only produces good measures if the spread and shape of the *Pareto-Front* are similar to the optimal *Pareto-Front*. Sayyad et al. (2013) state that *Hypervolume* is the most widely applied performance indicator, being applied in a total of 12 studies out of a reviewed pool of 51. However, none of the reviewed research efforts utilises *Hypervolume* in problem domains that are related with the present research.

### *3.3.5 Methods of Statistical Analysis*

The present research evaluates the feasibility of *MOEA* implementations in the problem domain of software modularisation and architecture reconstruction. The *MOEA* implementations that are applied in this research feature strategies that are of a probabilistic nature to overcome the limitation of deterministic optimisation approaches that can stall in local optima (compare section: 2.4.2). Hence, multiple executions of the same objective configurations are likely to discover different optimal *Pareto-Fronts* due to the employment of these probability-based characteristics.

Correspondingly, it is not feasible to derive conclusions on the performance of an employed optimisation configuration based on a single execution. Evaluation methods need to be implemented that address these probabilistic characteristics of the applied *MOEA* implementations and thus enable statements to be made regarding the general performance of algorithms and reconstruction configurations.

Arcuri and Briand (2011) analyse the methods and evaluation techniques of optimisation approaches in sixteen research papers that apply randomized algorithms in problem domains of software engineering. They highlight that some of the evaluation methods that are applied in the reviewed papers are insufficient to support thorough conclusions on the performance of the analysed problem implementations and algorithms (e.g. no application of multiple runs (8 papers) and/or no application of statistical testing (11 papers)). Based on the review, Arcuri and Briand (2011) suggest a set of practical guidelines for the thorough performance evaluation of randomised algorithms. The key elements of these practical guidelines are the application of multiple repetitions of runs, application of Mann-Whitney U-tests for non-parametric results or t-tests for result sets that feature normal distribution characteristics, reporting of all obtained p-values regardless of whether or not these exhibit significance, and reporting of standardized effect sizes. Additionally, means, standard deviations, variance min/max values, skewness, median and absolute deviation should be reported if possible to support the application of meta-analysis. Finally, where the evaluation considers more than two randomised algorithms, Arcuri and Briand (2011) suggest a sequence of pairwise comparisons of descriptive statistics followed by pairwise statistical tests and effect size measures.

### *Multiple Runs and Sample Size*

The probabilistic nature of the employed *MOEA* implementations demands the execution of multiple runs of metaheuristic configurations followed by the calculation of descriptive statistics. The literature on the evaluation of optimisation approaches gives different advice on an optimal number of reruns that are required to derive sound

assumptions on the performance of an optimisation setting. Talbi (2009) specifies a minimum of 10 runs using identical settings and postulates an optimum of 100 identical runs to support thorough statistical analysis. Rice (2007) claims that 30 reruns are sufficient to reach representative results. The results of these reruns create the populations from which descriptive statistic measures can be calculated. Certainly, a higher number of reruns enables more representative descriptive measures. However, the number of reruns of an objective setting is limited by technical feasibility and available resources.

Additionally, it needs to be considered that a bigger sample size increases the likelihood of the rejection of the null hypothesis if statistical hypothesis testing methods are applied. This effect can become problematic if too large sample sizes are used and marginal differences become statistically significant even if the differences themselves have no practical importance (Lenth, 2001). Hence, effect size and descriptive statistic measures need to be applied *in combination with* statistical hypothesis testing methods to enable reliable statements to be made on the difference in performance of different optimisation configurations.

The common approach in optimisation research is to conduct the analyses based on the final *Pareto-Front* of the analysed search configurations (Coello et al., 2007; Talbi, 2009). In the evaluation of the present research twelve reruns of an optimisation configuration have been conducted to create datasets that exhibit sufficient statistical value. Hence, an optimisation configuration that consists of a *MOEA* configuration setting (*MOEA* implementation, variation operator setting), a system that is analysed, a conceptual architecture configuration, objective setting and a reconstruction configuration is executed twelve times. Hence, the analyses of metaheuristic configurations without any agglomeration of solution sets at a particular time of the search feature at least a sample size of twelve individuals in each population.

The analysis without the agglomeration of solution sets (e.g. projects, architecture settings, *MOEA* implementation or variation operator) feature limited informative value so the agglomeration is a valuable approach to support the validity and feasibility of an objective setting. A multi-objective evaluation framework has been developed that enables the agglomeration of configuration aspects based on the desired analysis (compare: section 4.3). This agglomeration of data sets increases the sample size. For example, the number of samples increases to 50 (5 systems x 10 reruns) per algorithm (6) if an analysis of an architecture reconstruction scenario across all evaluation systems is conducted.

Additionally, the implemented evaluation framework extends previous research by featuring functionality to agglomerate and slice data sets by configuration attributes (compare: section 5.5) and functionality to include multiple snapshots in the analysis to analyse the speed of convergence of the search (compare: section 5.6). These analysis configurations have an impact on the sample size. Hence, agglomerating multiple configuration aspects to, for example, evaluate the performance of a MOEA implementation in different problem scenarios might lead to large sample sizes. Again, such large sample sizes are prone to create statistically significant results even if the effect size and differences in mean or median are minor. Hence, in particular in such large sample sizes it is important to consider the differences of descriptive statistics and effects size measure to enable thorough conclusions on the differences of the compared samples.

### *Distribution Characteristics*

The distribution characteristics of the samples are an important attribute to determine appropriate statistical methods that are conducted to analyse statistical differences in population sets. Different tests such as the *Kolmogorov–Smirnov*, *Shapiro-Wilk*, *Pearson's chi-squared* tests and others exist to determine if a population features normal distribution characteristics (Razali & Wah, 2011). As discussed in the previous section, the samples that are analysed in the evaluation of this research feature a range

of different sizes and can contain a large number of individuals. The *Shapiro-Wilk* test is able to determine normality for populations sizes of n >= 5 up to n <=2000 (Razali & Wah, 2011). Razali and Wah (2011) state that it is an established practise to apply the *Kolmogorov–Smirnov* test for sample sizes bigger than 50 individuals. Hence, in this research, the *Kolmogorov–Smirnov* test has been used to test if the populations feature a normal distribution. The null-hypothesis of the *Kolmogorov–Smirnov* is that a population is normally distributed (Razali & Wah, 2011). A p-value is calculated that indicates if a population departs from a normal distribution. A common threshold to reject the null-hypothesis of the *Kolmogorov–Smirnov* test and correspondingly to determine if a population features normal distribution is 0.05. Hence, a p-value of > 0.05 expresses that a distribution features normality and vice versa. A threshold of 0.05 is also applied in the present research to determine if the normality condition is fulfilled.

The application of the *Kolmogorov–Smirnov* test on the data sets that are analysed in the evaluation of this research revealed that the collected data sets do not follow the characteristics of the normal distribution (compare results Chapter 4). The circumstance that the normality condition is not fulfilled limits the application to non-parametric statistical methods.

### *Statistical Hypothesis Testing*

The presentation of descriptive statistics such as mean, median, standard deviation and range is useful to gain insight into the performance of the different optimisation configurations and enables statements to be made regarding differences in performance. However, the exclusive application of descriptive statistics is not sufficient to determine the level of confidence with which the differences between the populations might be claimed to be accurate. Statistical hypothesis testing can thus be conducted on the collected datasets to confirm if the data sets feature statistically significant differences. If normality conditions are fulfilled the most commonly applied statistical tool is the *student t-test* (Arcuri & Briand, 2011). Arcuri and Briand (2011) and Talbi (2009) suggest the application of non-parametric analysis methods, such as *Mann–*

*Whitney–Wilcoxon*, *Permutation Test*, *Bootstrap*, *McNemar*, *Friedman* and *Kruskal-Wallis,* if the normality condition is not fulfilled.

Given the non-normality of the data sets collected in this research, the *Mann–Whitney–Wilcoxon* test is applied as a non-parametric statistical tool to determine any statistical differences between individual pairs of data groups. The *Mann–Whitney–Wilcoxon* test reveals if a set of samples of metric measurements of metric M obtained by search configuration S1 is significantly different than a set of samples of metric measurements of metric M obtained by search configuration S2.

The present research compares the performance of different *MOEA* implementations and architecture reconstruction settings across multiple evaluation systems. As such, multiple *Mann-Whitney-Wilcoxon* tests need to be implemented to identify statistical differences between the individual optimisation settings. The application of multiple significance tests increases the risk of a false positive significant outcome, known as a Type I Error (Sheskin, 2003). For example, the performance of six algorithms is compared within the evaluation of this research. The individual comparisons of the performance of these algorithms requires the execution of 15 (6(6-1)/2) significance tests.

The procedure to counteract the problem of false positive significant outcomes is to conduct an upstream statistical test to reveal if statistical differences between the dataset groups exists, followed by the application of a post-hoc analysis to compare pairs of data-set groups to identify individual statistical differences between data set groups (Sheskin, 2003). Within this research the non-parametric *Kruskal-Wallis* and the parametric one-way *ANOVA* are applied to identify if any significant differences exist between data-set groups. The *Kruskal-Wallis* and one-way ANOVA return a significance level (p-value) that reports the probability that no difference between the data sets exists. A commonly accepted and widely applied threshold of the p-value to accept the rejection of the null hypothesis in empirical research is $p < 0.05$ (Sheskin, 2003).

If the application of the *Kruskal-Wallis* or one-way *ANOVA* confirms a significant difference in the data groups, a post-hoc analysis is applied to identify individual differences between the data set groups. Pairwise *Mann-Whitney-Wilcoxon* (non-parametric) or *student t-tests* (parametric) are applied to determine the statistical difference between the individual population groups.

However, the increased risk of Type I Errors remains through the application of multiple significance tests. A common practise to reduce the risk of false positive outcomes is to adjust the significance level of the applied post-hoc tests (Sheskin, 2003). The correction method that is applied in the evaluation of this research is the *Bonferroni-Dunn* method in which the desired significance level is divided by the number of comparisons (Dunn, 1961). The number of conducted pairwise significance comparisons can be determined by calculating the triangular number $T(n-1) = \frac{n(n-1)}{2}$ where $n$ is the number of populations. For the previous example, 6 algorithms are compared and a total of $T(n-1) = 15$ significance tests are required. Correspondingly, the corrected significance level is calculated as follows $\frac{0.05}{15} = 0.0033$.

### *3.3.6 Summary*

The previous sections presented the selection of architecture reconstruction scenarios, evaluations systems and *MOEA* implementations that are utilised in the experimental evaluation of the developed framework. Furthermore, methods are discussed that are crucial for the thorough analysis of the collected results. An outcome of this research is the implementation of an evaluation framework that embodies the described techniques and enables the comparative analysis of the performance of different metaheuristic tunings, *MOEA* algorithms and reconstruction configurations automatically. The implementation and application of this evaluation framework is presented in chapter 5.

## 3.4 *Communication of Findings*

This thesis presents the documentation of the research stages that contribute to the development of the suggested approach and the findings of the above stages. Additionally, to provide full traceability for the collected results and findings, the data is published on the project webpage[22]. Furthermore, the developed source code is also available on the project webpage to enable other researchers to validate and extend the work. The publication in research journals and conferences that address work in the area of software architecture reconstruction and search based software engineering is currently underway to make the findings available to the wider research community. These steps support the communication of the problem, its significance, the utility, novelty and rationale of the solution, the rigor of its design, and its demonstrated effectiveness, to other researchers.

[22] http://code.google.com/p/rearchitecturer

# 4 Design of a Multi-Objective Architecture Reconstruction Component

The objective of this research is to evaluate the feasibility of multi-objective optimisation strategies when applied in the area of architecture reconstruction to identify feasible architecture classifications that can operate as a starting point for the modularisation of software systems and the containment of software erosion. A novel approach has been developed to address relevant and contemporary challenges in the targeted problem domain. As highlighted in section 2.4.3, previous research that applied *SBSE* in similar problem domains features pre-defined sets of optimisation goals and *MOEA* configurations. This research proposes a more flexible design to enable the execution and evaluation of different configurations.

More specifically, the developed approach enables the stakeholder to define search configurations by orchestrating attributes of the configuration of metaheuristics, reconstruction objectives and optimisation goals. Additionally, this work integrates the notion of conceptual architecture models in the search process. Depending on the configuration, a conceptual architecture model can operate as a target design or be reconstructed as an objective of the search process. Furthermore, a set of candidate methods are suggested to enable the stakeholder to review, constrain and analyse the visited solutions. Additionally, the approach demonstrates the feasibility of seamless integration of discovered solutions into the architecture management and monitoring process.

Figure 21 depicts the process that has been developed in this research to first employ the search and then support the user in identifying a final solution from the pool of visited solutions in the multi-objective search process.

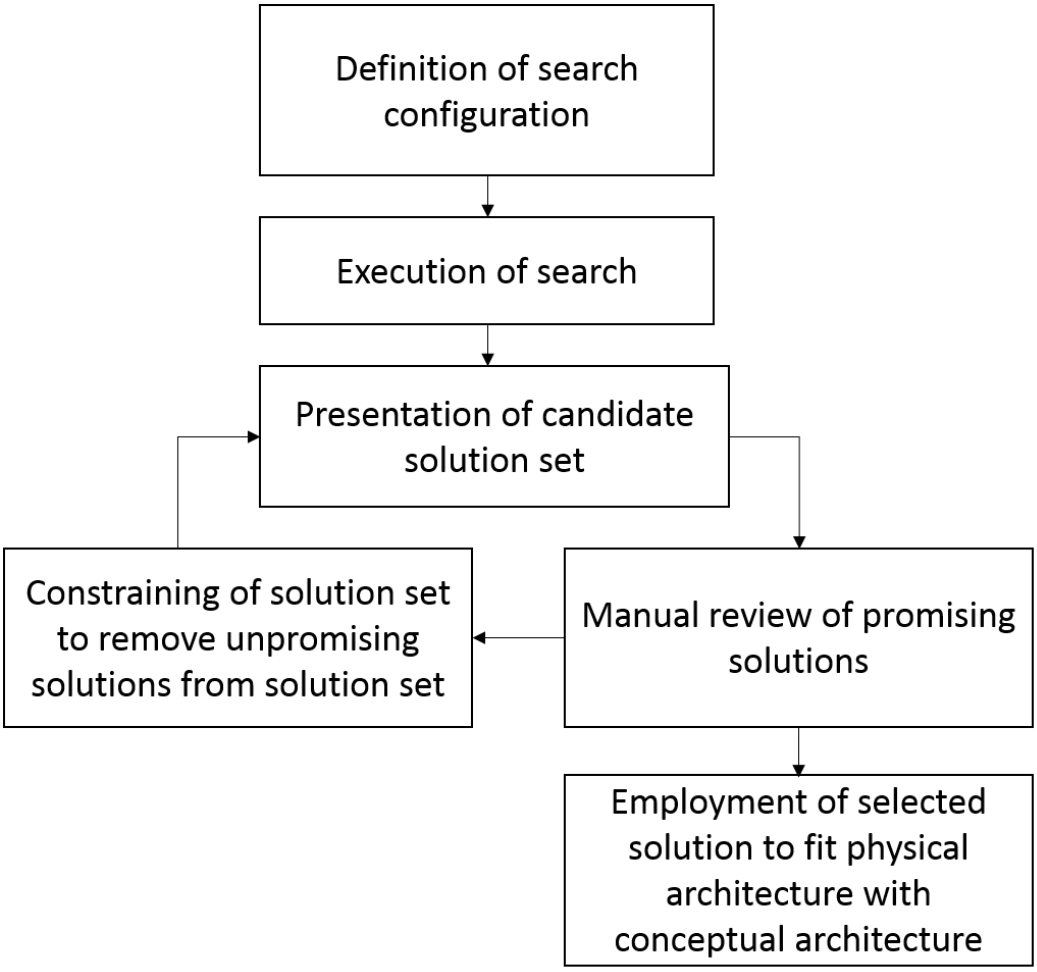


**Figure 21: Process of Multi-Objective Architecture Reconstruction Framework**

An open source artefact, called *Rearchitecturer*[23]*,* has been developed in the course of this research as a prototype implementation of the developed approach to enable the evaluation of the objectives of the present research.

The *Rearchitecturer* artefact enables the employment of a diverse set of state-of-the-art *MOEA* implementations and *MOEA* tunings. The employment of a variety of *MOEA* implementations and tunings can be overwhelming in practise and might well be beyond the capabilities of a 'normal' software designer who is likely to have a limited understanding of the concepts of multi-objective optimisation. Certainly, the practicality

[23] http://code.google.com/p/rearchitecturer/

of using such a flexible solution is disputable. However, little is known on the impact of different *MOEA* tunings in software engineering problem scenarios (see: section 2.4.3). Hence, at this stage no reliable recommendations can be made *a priori* regarding a *MOEA* tuning that performs well in the targeted problem domain. Correspondingly, the present research suggests an approach that allows the user to control a wide range of *MOEA* implementations and tunings to extend the body of knowledge of the performance of individual *MOEA* tunings in the targeted problem domain. Hence, the design of the implemented prototype needs to be understood as a prerequisite to enable the discovery of the impact on the performance of individual tunings for the later recommendations of pre-set tunings.

The next sections focus on the description of the tool's functionality and typical use of the *Rearchitecturer* artefact and describe how it addresses the targeted research objectives. Discussion of specific implementation details is kept to a minimum. Nevertheless, in some sections references to specific implementation classes are given. The source code of the *Rearchitecturer* artefact can be downloaded from the project webpage. Hence, if further information on the implementation is required the *Javadoc* of the corresponding classes is a good starting point.

The chapter is divided into two parts, section 4.1 presents the demonstration of the supported configuration aspects and section 4.2 describes the implemented methods to present and review search results. Furthermore, the *Rearchitecturer* artefact features the option to empirically evaluate multi-objective search results. The employment of the multi-objective evaluation framework is presented in chapter 5 as it is independent of the architecture reconstruction problem context.

## 4.1 *Rearchitecturer Search Configuration*

The configuration is a fundamental component for the effective execution of the search and the subsequent evaluation of the results. The search configuration defines the applied objectives, the reconstruction objectives, the *MOEA* configuration and information related to the physical and conceptual architecture.

The search configuration is represented by an XML file that defines the input directory of the software system's physical class files and an optional conceptual architecture model that can be used as a target design in the architecture reconstruction process. Details on the reasoning for the inclusion of an architecture description, and aspects of the structure of these architecture descriptions, can be found in section 4.1.1.

Another crucial component in the configuration of the search is the definition of the optimisation goals (see section: 4.1.2), reconstruction objectives (see section: 4.1.3) and the *MOEA* tuning (see section: 4.1.4). The search configuration is managed in an instance of the *experiments.SearchConfiguration* class. The *JAXB*[24] framework is utilised as a persistence framework to store and load *SearchConfiguration* instances.

The *Rearchitecturer* component features a *Swing* based *Graphical User Interface (GUI)* through which the user is able to define, edit and employ search configurations and architecture descriptions. The *GUI* is started by executing the main method of the *startup.RearchitecturerStarter* class. Figure 22 pictures a screenshot of the *Rearchitecturer* configuration perspective and highlights the visual components that are responsible for the definition and employment of the search configuration and architecture description.

[24] https://jaxb.java.net/

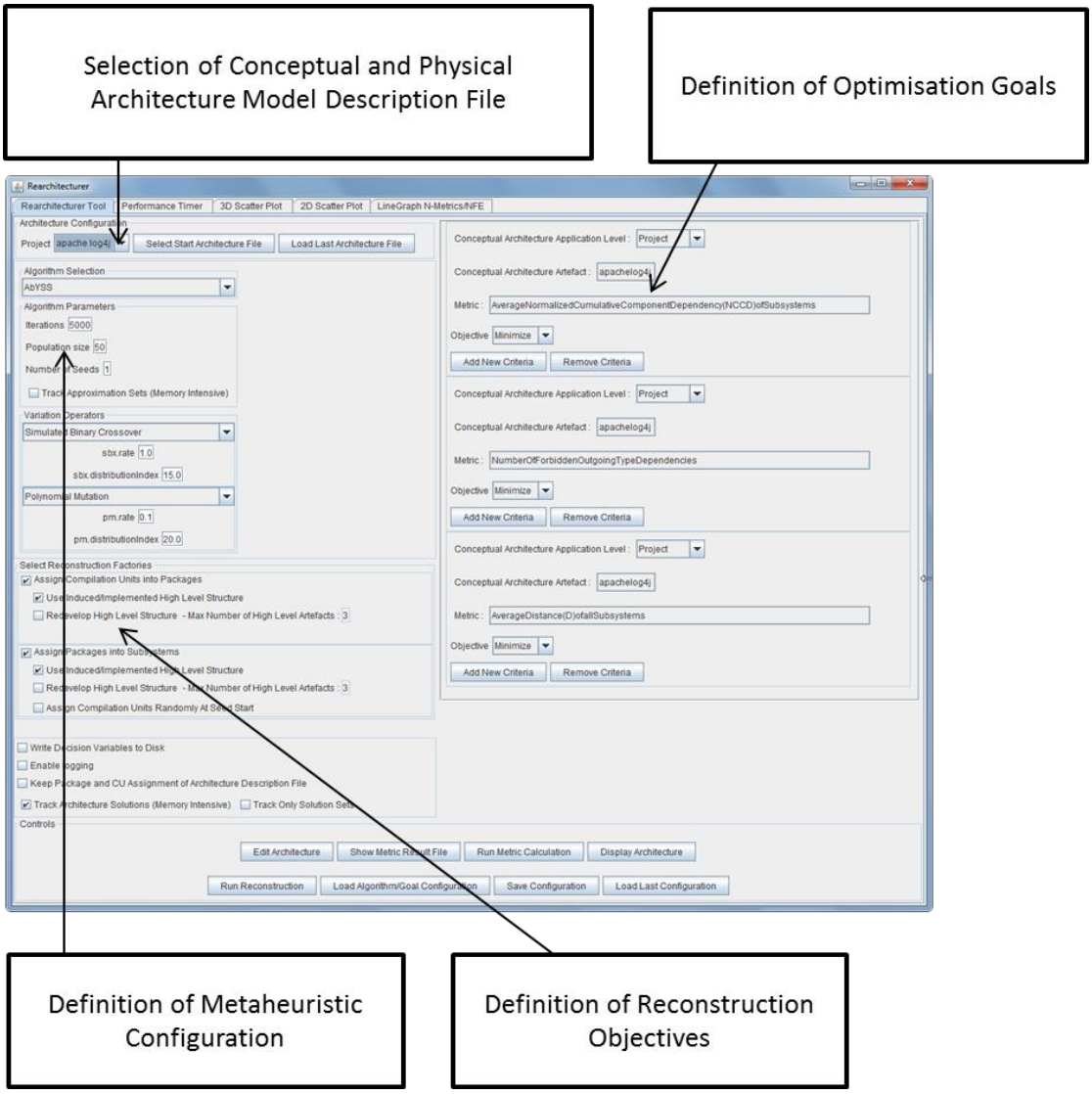


**Figure 22: Screenshot of the *Rearchitecturer* search configuration *GUI***

The configuration perspective enables the stakeholder to define a search configuration setting by selecting the conceptual and physical architecture, the optimisation goals, the metaheuristic configuration and the reconstruction objectives. The controls at the bottom of the configuration perspective of the *Rearchitecturer* artefact enable the initiation of the reconstruction based on the defined configuration. Additionally, the configuration perspective of *Rearchitecturer* enables the saving, loading and altering of reconstruction configurations. Additionally, the *Rearchitecturer* artefact comprises functionality that supports the employment of search configurations

and architecture descriptions in batch mode to enable the execution of different reconstruction and *MOEA* configurations (see section: 4.3). Furthermore, the architecture description and experiment configuration collectively operate as a description of the gathered datasets and are used in the empirical analysis framework to determine differences between datasets (see section: 4.3).

### *4.1.1 Definition of Conceptual and Physical Architecture*

To address the objective of this research an approach has been developed that supports the classification of software artefacts into high-level architecture descriptions based on the definition of multi-objective metric goals. The application of an architecture reconstruction approach implies that the conceptual architecture is at least partially lost and that it is not trivial to re-establish a conceptual architecture model of the software system (Koschke, 2008). Nevertheless, development stakeholders might have a partial understanding of the desired architecture design of the system. The integration of such information into the architecture reconstruction process can then lead the search to architecture configurations that facilitate desired dependency and modularisation characteristics. Consequently, the *Rearchitecturer* component supports the definition of a conceptual architecture configuration to facilitate the consideration of a conceptual architecture model during the modularisation process. Additionally, the design of the reconstruction objectives enables the discovery of instances of the conceptual architecture model during the search process (compare section: 4.1.3 for a more detailed description). Some of the implemented software design metrics, which can be utilised as optimisation goals, consider the conceptual architecture model (compare section: 4.1.2). The employment of such metrics as optimisation goals further enables the inclusion of the conceptual architecture model into the optimisation process.

The developed prototype supports the definition of subsystem- and layer-based conceptual architecture models. The conceptual architecture model can also entail layers and subsystems. Additionally, allowed and forbidden dependencies between

architecture artefacts can be defined. The implementation of a conceptual architecture model based on subsystems and dependencies between subsystems thus enables user-driven definition of most of the established high-level architecture patterns. The conceptual architecture model is defined in an XML-based architecture description file.

Furthermore, the entities of the physical architecture also need to be defined, to enable the classification of the physical architecture entities into the conceptual architecture model. As implemented, *Rearchitecturer* focuses on the modularisation of *java* systems. Hence, the input directories of the *java* class files need to be defined to inform *Rearchitecturer* which compilation units have to be parsed to build the physical architecture model. This definition of the location of the *java* class files is also included in the described XML architecture description file.

The format of the architecture description file, to define the conceptual architecture model and the location of the compilation units, follows the general structure suggested in the *Dependometer*[25] framework. The architecture description file can be loaded, saved and edited within the *Rearchitecturer* configuration frame or be included as part of the system call in the batch execution mode. Appendix B depicts an example of such an architecture description file. Other examples of architecture description files that have been utilised within the evaluation of the objectives of the present research are available on the *Rearchitecturer* project website.

### *4.1.2 Definition of Optimisation Goals*

The objective of this research requires the evaluation of the feasibility to apply search based software modularisation approaches in the architecture classification and reconstruction domain. Concepts of architectural design need to be included to evaluate the applicability of *SBSE* in architecture reconstruction scenarios. Both Aleti, Buhnova, Grunske, Koziolek, and Meedeniya (2013) and De Silva and Balasubramaniam (2012)

[25] http://sourceforge.net/projects/dependometer/

describe the grouping of individual elements into components of higher abstraction levels as one of the fundamental concepts of software architectures. Additionally, the usage of a conceptual architecture models is an established method to assess and monitor the erosion of software systems (De Silva & Balasubramaniam, 2012).

The present research applies the proposal of Harman and Clark (2004) to utilise software metrics as fitness functions within optimisation approaches. Section 2.4.3 illustrated other research that has utilised software metrics as fitness functions in the domain of search based driven software modularisation and clustering. The present research states that the objective settings of previous approaches might not be suitable to reconstruct architecture configurations as the employed objectives potentially conflict with recognised principles of software architecture design (e.g. minimizing the number of inter-edges, *MQ*). Certainly, it is an established principle of good software engineering to design solutions that feature low coupling. However, as noted previously, dependencies between modules should not necessarily be seen as being unwanted. For example, dependencies that align with the desired dependency flow of the conceptual architecture should not be penalised. Additionally, the described approaches suggest a fixed set of objectives and define these as the means by which to reconstruct software modularisation designs.

The present research contributes a multi-objective architecture reconstruction approach that includes specific architecture design metrics more suited to assessing the quality of an architecture configuration.

Software design metric considers quality aspects at a specific abstraction level. For example, a software design metric might operate on the micro-design level and hence only evaluate aspects within a method or compilation unit. However, other metrics operate on higher abstraction levels and examine quality aspects within the dependency structure of a set of compilation units. Metrics that measure aspects of high abstraction

levels are particularly relevant for the present research due to the explicit focus on reconstructing high-level architecture models of software systems.

This research asserts that the integration of a conceptual architecture model is a beneficial approach to guide the search towards desired modularisations. Instead of developing such functionality from scratch, the *Dependometer* architecture conformance checking engine is integrated into the *Rearchitecturer* artefact to parse the conceptual architecture model and enable the calculation of some of the implemented metrics. The employment of the *Dependometer* artefact enables the inclusion of conceptual architecture models that feature subsystem, horizontal and vertical layer-based architecture models. The *Dependometer API* supports the definition of forbidden and allowed dependencies between the elements of the conceptual architecture model. Nevertheless, changes to the *Dependometer* code-base have been developed in this research to enhance the performance of the *Dependometer* architecture verification process to enable its application within a *SBSE* approach. Additionally, the *Classcycle* library is utilised instead of the built-in *Dependometer* parser to build the physical architecture model of the system.

Overall the developed approach supports a wide range of software architecture metrics that consider the defined conceptual architecture. The *Rearchitecturer* artefact features, in the current implementation, a total of 57 project-, 23 layer- and 21 subsystem-metrics that express quality attributes as numerical values. The relevance and background of the main architecture design metrics (cohesion, coupling, architecture violations, cycles and structure assessment metrics) that are employed in the evaluation of this research have been addressed in detail in section 2.3. Nevertheless, a variety of additional metrics have been implemented in the *Rearchitecturer* artefact. However, while potentially of interest to those designing and refactoring systems, these metrics are of secondary relevance in the evaluation of the objective of the present research and a detailed description is beyond the scope of this thesis. A list of the implemented metrics can be found on the *Rearchitecturer* project page; all of the metrics are derived

from existing literature and this research is not proposing new metrics. The cycle metric implementations have been implemented from scratch as no suitable open-source implementation could be found that featured suitable performance. The implementation of the cycle metrics to determine the number of cycles on layer, subsystem and package levels is based on the algorithm presented in Tarjan (1972) that describes an approach to determine the number of strongly connected components in a dependency graph.

The Rearchitecturer component supports stakeholders in their defining a set of optimisation goals in a flexible manner to express the desired architectural design of the system. The defined optimisation goals operate as objectives in the employed search. Hence, the optimisation goals are a representation of the desired architectural design of the system. A diverse configuration of design objectives is supposed to guide the search towards solutions that feature good and balanced quality in the desired objectives. Figure 23 depicts the *GUI* component that represents an optimisation goal configuration within the *Rearchitecturer* configuration perspective. The depicted optimisation goal targets a value of 1.0 of the *Normalized Cumulative Component Dependency (NCCD)* metric in the subsystem *level5::application*.

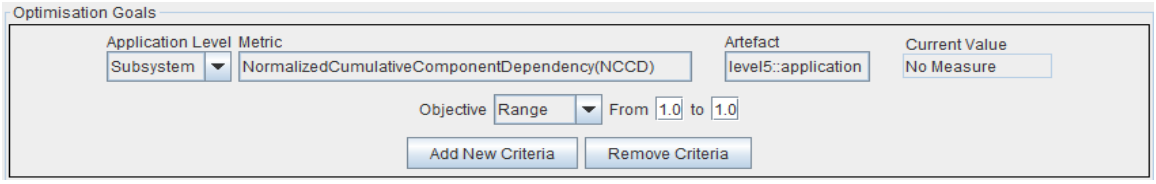


**Figure 23: Optimisation goal configuration**

An optimisation goal consists of one of the implemented software design metrics, an application level and a desired optimisation objective. Additionally, a specific conceptual architecture artefact has to be selected if the application level is on a layer, subsystem or package level. Furthermore, a desired optimisation direction or targeted range has to be defined. Possible optimisation objectives are the minimisation and maximisation of the selected software design metric or the definition of a desired range of the design

metric measurement that should be evident in the solutions. The range optimisation objective requires the definition of a “from” and “to” value.

Harman, Clark, and Cinnéide (2013) and Nakib and Siarry (2013) highlight that the evaluation of performance is crucial for the successful application of search based approaches. Hence, development activities have also been conducted in the present research to optimise the performance of the applied framework. For example, development activities have been conducted to optimise the metric calculation process to avoid unnecessary computational overhead. According to the utilised research methodology the effect of such development activities on the performance has been iteratively reviewed to enable a targeted implementation of the research objectives.

### *4.1.3 Definition of Architecture Reconstruction Objective*

Architecture reconstruction is a type of reverse engineering in which architectural information is reconstructed for an existing system. This information can be gathered from the source, the system execution, available documentation, stakeholder interviews and domain knowledge (Koschke, 2008). As presented in section 2.2, multiple architecture views onto a software system exist. The focus of the present research is to reconstruct and discover views of the modularisation and implementation perspective of software systems based on the analysis of the source code.

The present research contributes a dynamic modularisation approach that enables the reconstruction of architecture classifications on different abstraction levels. The implemented reconstruction strategies support the classification of low level artefacts (compilation units or packages/folders) into abstraction artefacts of the next higher abstraction level (packages/folders or subsystems). Hence, the reconstruction strategies that can be employed in the *Rearchitecturer* component are:

- The classification of compilation units into packages/folders
- The classification of packages/folders into subsystems

The implemented reconstruction strategies can be employed individually or agglomerated to classify compilation units and packages/folders conjointly in the corresponding high-level artefacts.

Additionally, the reconstruction strategies feature, besides the classification of low-level artefacts into the corresponding high-level artefacts, the optional development of the corresponding high-level artefacts e.g. packages/folders and/or subsystems and subsystem dependencies. The classification strategy classifies low-level artefacts into the corresponding developed high-level artefacts if the creation of high-level artefacts is employed. Constraints can be set to define the maximum number of packages and/or subsystems depending on the employed classification strategies. If the re-development of the abstraction level is not aspired the implemented package/folder structure or defined conceptual architecture model definition is utilised as the default partitioning for the classification of low-level artefacts. The reconstruction objective can be defined in the configuration perspective of the *Rearchitecturer artefact.* Figure 24 depicts the *GUI* of the reconstruction objective configuration.

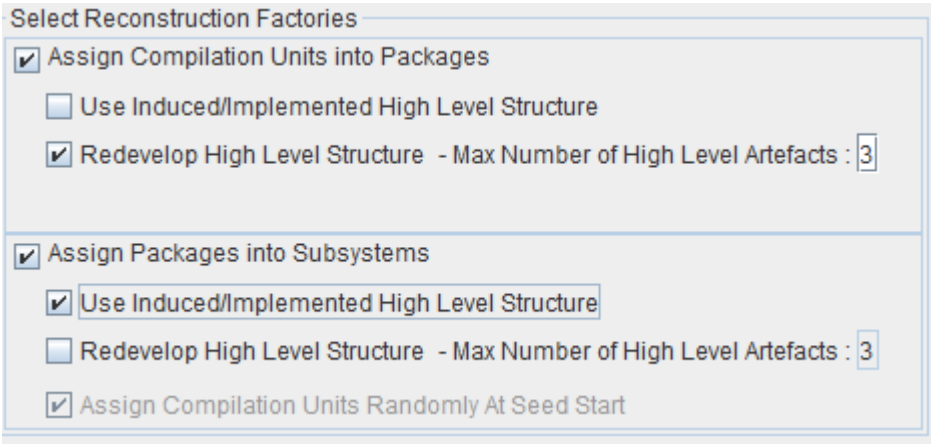


**Figure 24: Reconstruction goal configuration**

The reconstruction objective is the key component to determine the problem representation. The reconstruction objective is represented by the definition of the type and size of the decision variable and the implementation of the encoding. As depicted in

section 4.3.1 and implemented within this research, the encoding transforms a decision variable instance, which is created by the employed *MOEA* algorithm, into an architecture modularisation instance.

The problem representation is implemented as a set of decision variables that are defined based on the physical and conceptual architecture model and the reconstruction objective. Consequently, the genotype representation as the number, type and range of the decision variables depend on these configuration parameters. The genotype instances are transformed into a phenotype representation that allows the calculation of the implemented software design metrics. The transformation process also considers the configuration parameters to create a valid phenotype instance. The phenotype representations are implemented as different graph models that represent the required aspects of the physical and conceptual architecture model and enable the efficient calculation of the implemented metrics. Additionally, the *Rearchitecturer* component features functionality to export phenotype instances into an architecture description file. The exported architecture description file can be utilised by the *Dependometer* architecture management framework. Hence, the generated architecture description file can be employed as a conceptual architecture model within the architecture management and monitoring process. The following sections discuss aspects of the design and implementation of the two implemented reconstruction strategies in more detail.

### *Classification of Compilation Units into Packages/Folders*

The classification of compilation units into subsystems strategy supports the classification of compilation units into the existing or a re-developed package/folders structure. The design of the encoding is implemented as a set of integer variables. The number of integer variables is defined by the number of compilation units in the analysed software system. Each of these compilation unit decision variables features a range. The range is defined as an integer value between zero and the number of packages in the system. A genotype instance is created from the employed *MOEA* by

outfitting all decision variables with integer values in the defined range. Each of the genotype instances that are created by the *MOEA* represents a different classification of compilation units into packages/folders. If the re-building of the package structure is selected the maximum number of packages, which is defined by the stakeholder, is utilised as the upper end of the range for each of the compilation unit decision variables. A package is created in the conceptual architecture model representation (phenotype) if at least one compilation units has been assigned to the package based on the genotype representation.

Another aspect that is relevant for the application of *SBSE* is the complexity of the solution space. The total number of solutions of the assigned compilation unit reconstruction strategy that can be visited depends on the number of compilation units and the number of package artefacts. The number of all possible solutions based on the illustrated design of the decision variables for a system *S* with a number of compilation units *C* and a number of packages *P* can be calculated as follows:

$$Complexity\ (Classify\ Compilation\ Units)\ S = P^C$$

The complexity function illustrates that the classification of compilation units features exponential complexity based on the number of compilation units and a non-trivial package structure. As depicted in section 3.3.3, this research utilises a set of open source systems to evaluate the feasibility of the developed approach. It is straight-forward to calculate the number of possible solutions based on the compilation units and package information of the evaluation systems that are given in Table 1 in section 3.3.3. Even the smallest of the evaluation systems (Log4j v.1.2.17) features a number of possible solutions of $24^{314} \approx 2.43 \times 10^{433}$. This high complexity of the total search space renders an exhaustive evaluation as infeasible. However, the high complexity also indicates that the application of *SBSE* is a justifiable approach to find reasonable good compilation unit partitionings.

### *Classification of Packages into Subsystems*

The classification of packages into folders reconstruction strategy supports the classification of packages into the subsystems of a defined or developed conceptual architecture model. Commercial and open-source architecture monitoring systems as for example *Sonargraph*[26], *Dependometer*[27], *Struture101*[28], *XRadar*[29] and *Lattix*[30] enable the mapping of the physical implementation by allowing stakeholders to assign packages into subsystems of a given conceptual architecture model. Hence, the implemented reconstruction strategy models this common use-case of software architecture management systems to enable stakeholders to assign packages or folders into subsystems of a conceptual architecture model.

The decision variable implementation is similar to the decision variable implementation of the classification of compilation units into packages reconstruction strategy. The main differences are that the decision variables represent packages instead of compilation units and the genotype decision variable values link to subsystems of the conceptual architecture model. Additionally, decision variables are required to enable the *MOEAs* to model subsystem dependencies if the conceptual architecture model is rebuilt.

A set of integer package decision variables represent the packages of the system. The package decision variables are altered by the employed *MOEA*. The value of the package decision variable represents the classification of a package into a subsystem. The pre-defined range of the package variables is limited to an integer value between zero and the number of subsystems within the conceptual architecture model. Hence, the genotype instance indicates the assignment of a package into a particular package. Additionally, the reconstruction strategy is able to rebuild the conceptual architecture

---

[26] http://*www.hello2morrow.com*
[27] http://source.valtech.com/display/dpm/Dependometer
[28] http://*structure101.com*
[29] http://*xradar.sourceforge.net*
[30] http://www.lattix.com

model. The stakeholder can define the maximum number of subsystems. The number is utilised as the upper end of the range for each of the package decision variables. A subsystem is created in the conceptual architecture model representation (phenotype) if at least one package has been assigned to the subsystem based on the genotype representation.

Additionally, if the conceptual architecture model is rebuilt, it is also required that the allowed dependencies between subsystems are defined. A matrix (number of maximum subsystems x number of maximum subsystems) of binary decision variables is employed as an encoding to re-build the allowed dependencies between subsystems. An allowed dependency is created within the phenotype if the binary decision variable is true and both subsystems contain at least one package. A drawback of this encoding implementation is that it allows the building of cyclic dependencies at the subsystem level. A remedy to encounter the creation of cyclic dependencies is the definition of an optimisation goal that penalises the creation of solutions that feature such unwanted cyclic dependencies e.g. minimise cyclic dependencies between subsystems or packages.

As this reconstruction strategy operates on a higher abstraction level with more coarse grained artefacts it is likely that the complexity to visit and evaluate all possible solutions is considerably reduced. Hence, it is even more pertinent to evaluate the complexity of the classification of packages into subsystems reconstruction strategy to determine if the degree of complexity justifies the application of *SBSE*. The complexity of the classification of packages into subsystems depends on the number of packages and the number of subsystems of the conceptual architecture model. The complexity of the classification of packages into subsystems and the rebuilding of the subsystem dependencies for a system S with a number of packages P and a number of subsystems N is calculated as follows:

$$Complexity\ S\ (Classify\ Packages) = N^P\ \times\ N^2$$

Even if the problem solution works with more coarse grained artefacts, the problem features an exponential complexity based on the number of packages and non-trivial subsystem configurations. To calculate the complexity of the classification of packages reconstruction strategy, a conceptual architecture model is needed, unless the conceptual architecture model is built as part of the classification process. Within the evaluation of this research conceptual architecture models are utilised with at least four subsystems. As illustrated in section 3.3.3 the smallest system that is analysed in the evaluation is Log4j (v.1.2.17) which comprises a total of 24 packages. The complexity of all possible package configurations of the *Log4j* system into a conceptual architecture model with four subsystems and rebuilding of the subsystem dependency structure is $4^{24} \times 4^2 \approx 4.50 \times 10^{15}$. This example shows that the building of subsystem dependencies decision variables adds a rather small degree of complexity to the overall complexity of this reconstruction strategy. Nevertheless, this example also illustrates that the complexity of the classification of packages into subsystems is too high to justify an exhaustive evaluation of all possible solutions in an acceptable amount of time. Instead *SBSE* seems to be a justifiable approach to find feasible package partitionings.

### *4.1.4 Metaheuristic Configuration*

Coello et al. (2007) and Talbi (2009) suggest a separation of *SBSE* implementations into three components: implementation of the metaheuristic, representation of a solution and the fitness function. This research adapts the suggested separation into these three components. The fitness function and problem representation are domain specific to exploit domain specific knowledge. As illustrated in section 4.1.2, the fitness function within the present research is implemented as an agglomeration of optimisation goals that are defined by the stakeholder. The design of the representation that has been illustrated in detail in section 4.1.3 is implemented as a set of decision variables in which number, type and range are adapted to the reconstruction objective, the analysed system and the aspired conceptual architecture model. The metaheuristic

implementation generates instances of the representation and employs the defined fitness function to evaluate the fitness of the generated representation instance. Parent selection, recombination, mutation and survivor selection strategies are employed by the applied metaheuristic implementation to evolve the fitness of a population. This decoupling of the metaheuristic implementation from the fitness function and representation implementation enables the re-use and flexible exchange of acknowledged metaheuristic implementations. This enables a straightforward comparative evaluation of different metaheuristic implementations.

The next section illustrates the employment of metaheuristics in the present research and the means that are available to stakeholders to control the selected metaheuristic implementation. The objective of the present research focuses in particular on the applicability of *MOEA* in the problem domain of architecture reconstruction and software modularisation. As described in section 2.4.3, related research so far has evaluated the applicability of *Hill Climbing* and *GA* metaheuristic implementations in the problem domain of software modularisation (Mitchell & Mancoridis, 2008; Praditwong et al., 2011). A much broader range of acknowledged *MOEA* implementations e.g. *Scatter Search*, *Particle Swarm Optimisation*, decomposition and differential evolution based *MOEA* implementations exist. Hence, it has been an objective of this research to enable the application of a diverse range of *MOEA* implantations.

A variety of libraries exists to support the application of metaheuristics in new problem domains. It has been decided to utilise existing metaheuristic frameworks within this research to access established *MOEA* implementations instead of implementing a set of proprietary and self-implemented adaptions of metaheuristic algorithms. Correspondingly, a part of the research effort within the present research has been the review of existing and applicable metaheuristic frameworks.

The *Watchmaker Framework*[31] is an open source *Java* library that exclusively targets single objective optimization. As outlined in previous sections of this thesis, the application of single objective optimisation is not considered to be sufficient to offer a comprehensive solution to the problem of architecture reconstruction. *ECJ*[32], *JMetal*[33], *MOEA*[34], *Opt4j*[35] *and PISA*[36] are multi-objective evolutionary algorithm optimisation frameworks. All of these frameworks feature advantages and disadvantages and likely all of them would have been applicable in the targeted problem domain. However, the present research utilises a combination of the *MOEA*, *JMetal* and *PISA* metaheuristic frameworks to enable the *Rearchitecturer* component to feature a wide range of different *MOEA* implementations. These three *MOEA* libraries have been selected as they offer a variety of different algorithms and the encoding model features good compatibility, so that its integration into the *Rearchitecturer* artefact has been relatively straightforward. Nevertheless, some minor extensions to the *PISA* and *JMetal* frameworks had to be made to be able to integrate the libraries in the developed component. These extensions mainly concern areas of parallel evaluation of objective functions, slight modifications of aspects of the encoding representation and modifications to enable the seamless support of exchangeable variation operators across the three *MOEA* frameworks. The *MOEA*, *JMetal* and *PISA* frameworks offer a total of twenty-eight multi-objective algorithms, and ten of these algorithms are implemented within the *Rearchitecturer* component. The selection ensures that a wide range of different evolutionary algorithm implementations are available without integrating duplicates or only slight variations of the *MOEA*, *JMetal* and *PISA* algorithms. Nevertheless, the integration of additional algorithm implementations of the *MOEA*, *JMetal* and *PISA* libraries is straightforward and can be easily conducted based on the

[31] http://watchmaker.uncommons.org
[32] http://cs.gmu.edu/~eclab/projects/ecj
[33] http://jmetal.sourceforge.net
[34] http://www.moeaframework.org
[35] http://opt4j.sourceforge.net
[36] http://www.tik.ee.ethz.ch/pisa/?page=selvar.php

current integration classes in the *Rearchitecturer* implementation. Table 2 depicts the algorithms that are currently integrated in the *Rearchitecturer* component, a classification of each algorithm and the reference that presented the implemented *MOEA* originally.

**Table 2: Implemented Evolutionary Algorithms in Rearchitecturer Component**

| **Algorithm** | **Algorithm Type** | **Reference** |
|---|---|---|
| AbYSS | Hybrid: scatter search + genetic operators | (Nebro et al., 2008) |
| CellDE | Hybrid: cellular genetic algorithm + differential evolution | (Durillo, Nebro, Luna, & Alba, 2008) |
| OMOPSO | Particle swarm optimization | (Sierra & Coello, 2005) |
| GDE3 | Differential evolution | (Kukkonen & Lampinen, 2005) |
| MOCell | Cellular genetic algorithm | (Nebro, Durillo, Luna, Dorronsoro, & Alba, 2007) |
| *MOEA*/D-DE | Decomposition based evolutionary algorithm | (Li & Zhang, 2009) |
| NSGA-II | Genetic algorithm | (Deb et al., 2002) |
| SMPSO | Particle swarm optimization | (Nebro et al., 2009) |
| SPEA2 | Genetic algorithm | (Zitzler et al., 2001) |
| Random | Random based evolutionary algorithm | |

The configuration perspective of *Rearchitecturer* offers support to select an algorithm, and to configure algorithm parameters and variation operators. Figure 25 depicts the visual implementation of the algorithm configuration component.

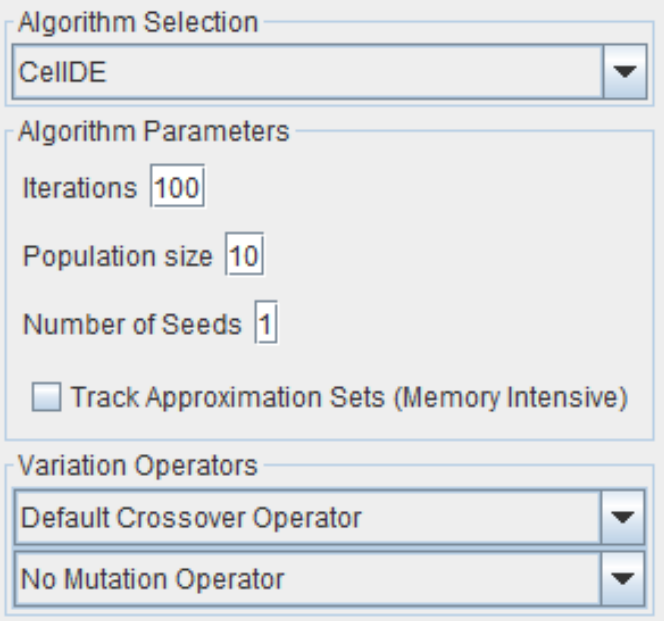


**Figure 25: *Rearchitecturer* metaheuristic configuration component**

The metaheuristic configuration component enables the user to select one of the implemented *MOEAs* depicted in Table 2. Furthermore, the number of iterations, the population size, the number of seeds and a crossover and a mutation operator can be defined. A seed is an independent run of the *MOEA* configuration in the applied problem. The number of seeds defines the number of runs that are conducted. The number of iterations determines the exit condition for each seed. An iteration cycle comprises the generation and evaluation of a solution. The population determines the number of solutions in a generation. All of the solutions of a generation need to be created and evaluated before the parent selection, recombination, mutation and survivor selection can be executed. Hence, a bigger generation gives the recombination strategies access to a wider gene pool. On the other hand, a smaller population features a more frequent application of parent selection, recombination, mutation and survivor selection mechanisms, but a smaller gene pool from which to create offspring.

All of the implemented *MOEA* algorithms exhibit some kind of probability-based strategies to overcome local optima. Consequently, an algorithm that runs with the same configuration is likely to result in different optimal *Pareto-Fronts*. Hence, multiple runs with identical configurations have to be conducted to enable the thorough evaluation and comparison of search results on which generalizable conclusions on the performance of an algorithm configuration can be based.

A range of crossover and mutation operators can be applied to change the recombination and mutation behaviour of the applied algorithm. *Simulated Binary Crossover*, *Differential Evolution*, *Parent-Centric Crossover* and *Unimodal Distribution Crossover* are the available recombination operators. *Polynomial-* and *Uniform-Mutation* are the available mutation operators that are implemented within the *Rearchitecturer* component. In general, it has been aspired to integrate each of the available variation operators seamlessly with the implemented algorithms. Nevertheless, some of the algorithms feature restrictions for the application of the variation operators. For example, the random search is by definition unable to utilise any of the variation operators. Additionally, some integration issues have been encountered during the implementation process so an immaculate operation of any algorithm and variation operator configuration is not guaranteed. Nevertheless, a detailed illustration of the conflicts of the supported *MOEA* algorithms and variation operators is not presented as part of this research. A detailed illustration about the incompatibility of the metaheuristic algorithm implementations and the variation operators would exceed the scope of this work and doubtfully add ample value to address the objectives of this work, as the variation operator are of secondary relevance during the evaluation process.

## 4.2 *Presentation, Constraining and Review of Architecture Design Configurations*

Previous research in related problem domains, such as that presented by Mitchell and Mancoridis (2008), Praditwong et al. (2011) and Abdeen et al. (2009), focuses on the identification and evaluation of one final best solution. The present research agrees with the general objective of previous studies that development stakeholders likely desire to identify an “optimal” solution, particularly given that one of the emphases of the presented approach is to create an architecture configuration that could be used in the architecture management and monitoring process of a software project.

However, it is also anticipated in this research that it is unlikely that the automatic identification and presentation of one “optimal” solution, that features the ”best” metric goal measurements, will in fact deliver a satisfying solution for every optimisation and application scenario. This is particularly infeasible as the present research employs multi-objective search techniques. The application of multi-objective approaches leads to an optimal *Pareto-Front* that most likely includes, for non-trivial objective configurations, multiple solutions (compare section: 2.4.2). Hence, the automatic selection of one single solution from an optimal *Pareto-Front* is not only infeasible but also impractical as it would require the discarding of promising solutions that dominate areas of the objective solution space. On the other hand it is also acknowledged that the presentation of printouts of approximation sets and *Pareto-Fronts* might be of little help to the development stakeholders who are seeking to identify an optimal solution or identify attributes that characterise a good solution. This implies that candidate techniques are needed to facilitate flexible support to development stakeholders to visualise approximation sets of software modularisation solutions and that stakeholders require functionality to review specific solutions in problem domain specific perspectives to identify promising solutions and/or solution aspects.

The implemented *Rearchitecturer* component therefore provides a set of candidate methods to visualise, review and constrain solution sets to enable the user to efficiently identify solutions that can be employed in the architecture monitoring and management process.

### *4.2.1 Presentation of Solution Sets*

The *Rearchitecturer* component features functionality to visualise different solution set configurations in *2D* or *3D* representations. The *Pareto-Front*, all visited solutions or a selected generation of solutions, can be visualised. The visualisation is not limited to the set of objectives that are configured in the search. Hence, the visualisation perspectives enable stakeholders to assign any of the recorded software design metrics (compare: section 4.1.2) to any axis in the visualisation perspective.

Figure 26 depicts a screenshot of the implementation of the *2D* solution visualisation component and highlights the sub-components that enable the stakeholder to review and constrain solution sets.

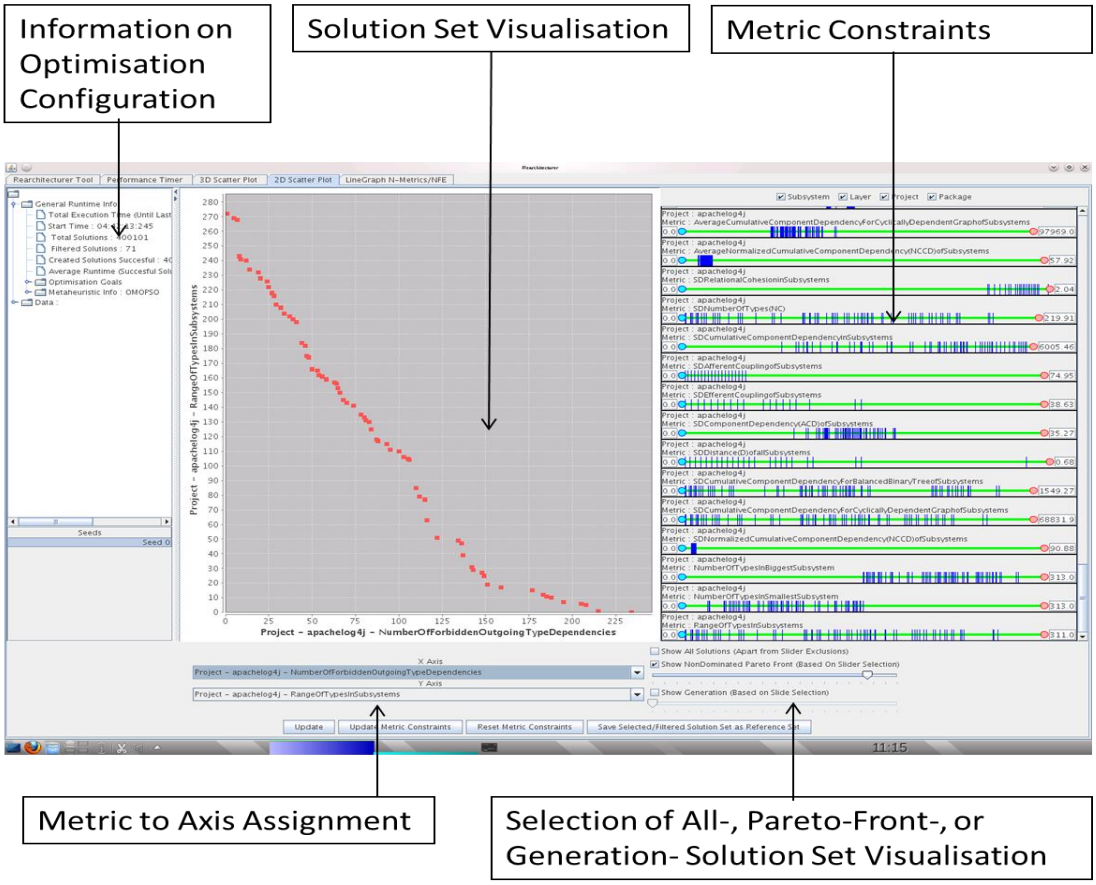


**Figure 26: Screenshot *Rearchitecturer* 2D solution visualisation**

The screenshot in Figure 26 has been created based on a search configuration with a maximum of 400,000 solution iterations and a population size of 100. Apache-log4j-1.2.17 is utilised as the evaluated system and the classification is conducted based on the assignment into a conceptual architecture model with four transient layers. The two objectives 'minimise the *Range Of Types In Subsystems* metric' and 'minimise the *Number of Forbidden Type Dependencies metric'* have been employed as optimisation goals in this search. This models the objectives of stakeholders to create solutions that feature no or very few architecture violations (represented by the minimise *Number Of Forbidden Type Dependencies* objective) and a fairly homogenous distribution of the number of types in the individual subsystems (represented by the minimise *Range Of Types In Subsystems* objective).

Additionally, the two reconstruction strategies *assign packages into subsystems* combined with *assign compilation units into packages* are applied. *OMOPSO (particle swarm optimisation)* is employed as a search algorithm. Each modularisation solution of the displayed solution set is represented by a corresponding entity in the coordinate system.

The metric to axis configuration enables the assignment of software design metrics to the individual axes of the solution set visualisation component. Figure 26 depicts the *optimal Pareto-Front* of the two objectives that are employed in the search on the *X*- and *Y*-axis at the end of the search.

The assignment of the metrics to the axes is not limited to the objectives of the search. Any recorded metric can be assigned to any of the axes. This allows the visualisation of solution sets from different software design perspectives. In the presented example, the *optimal Pareto-Front* features 62 solutions that dominate the other solutions that have been visited during the search. The 2D visualisation utilises the *JFreeChart* [37] library to visualise the solutions of a solution set.

The *3D* visualisation perspective features the same functionality as the *2D* visualisation with the main difference that an additional metric dimension is visualised. Figure 27 depicts an example of the 3D visualisation that features the following objectives: minimise *Number Of Forbidden Type Dependencies*, minimise *Normalised Cumulative Component Dependency (NCCD)* and minimise *Range Of Types In Subsystems*. The search features a maximum number of 200,000 iterations and the optimal *Pareto-Front* includes a total of 230 solutions. The other search configuration attributes are identical with the configuration presented in the previous example.

[37] http://www.jfree.org/jfreechart/

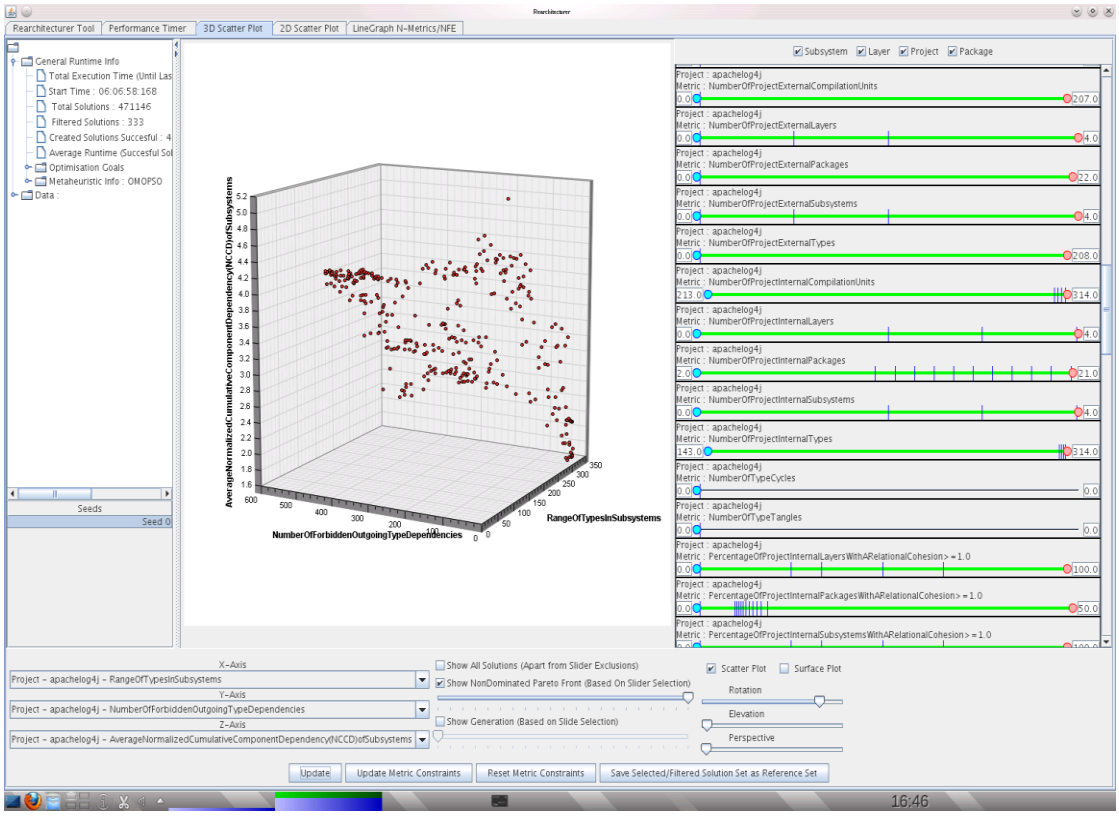


**Figure 27: Screenshot *Rearchitecturer* 3D visualisation**

This example models the objective of stakeholders to create solutions that feature no or very few architecture violations (minimise *Number Of Forbidden Type Dependencies*), a good distribution of types in the individual subsystems (minimise *Range Of Types In Subsystems*) and a structure within subsystems that features aspects similar to the structure of a balanced binary tree (minimise *NCCD*). The other search configuration attributes are identical with the configuration presented in the previous example.

The optimal *Pareto-Front* presented in Figure 27 indicates that the assessment of the objective attributes is challenging in a 3D visualisation. Hence, functionality is implemented to rotate and elevate the displayed 3D coordinate system to observe the solution set from different perspectives. The *ChartDirector* library is utilised to implement the 3D visualisation.

### *4.2.2 Constraining of Solution Sets*

The application of multiple objective optimisation results in *optimal Pareto-Fronts*. The solutions of these optimal *Pareto-Fronts* dominate areas of the objective space. However, the solutions of the optimal *Pareto-Front* are likely to feature different degrees of feasibility from the stakeholder perspective. Hence, stakeholders desire approaches to efficiently identify highly feasible solutions that can be employed in the target problem domain. Furthermore, the present research suggests that other promising solutions might be discovered that are not included in the *optimal Pareto-Front*. However, the inclusion of the complete set of visited solutions in the decision-making process renders the identification of a feasible solution as even more complex.

As has been noted previously, a variety of multi-objective approaches are implemented to tackle problems in software engineering domains (Sayyad & Ammar, 2013). The review of these approaches suggests that to this date no approaches exist that support stakeholders in the process of identifying highly promising solutions in multi-objective optimisation scenarios. A candidate implementation is suggested in this research that enables stakeholders to include software design and domain knowledge in the solution process to constrain a solution set to a manageable size and to identify highly promising solutions. The remainder of this section discusses relevant aspects to justify the implemented approach.

A result of the application of the concept of non-dominance is that optimal *Pareto-Fronts* often feature solutions that excel in a subset of the objectives and feature poor performance in other objectives. This is fine in most multi-objective optimisation problems as the solutions that are included in the optimal *Pareto-Front* are often highly competitive. However, it is anticipated in this research that some of the solutions that are included in the optimal *Pareto-Front* are not desirable from a software engineering perspective. For example, stakeholders most likely prefer software architecture compositions that feature a good balance between the objectives of the search. Hence,

it is unlikely that stakeholders would accept high trade-offs in some objectives to gain a particularly good outcome in another objective.

For example, the solutions from the previously visualised optimal *Pareto-Front* examples (Figure 26 and Figure 27) that feature a high Number Of *Forbidden Type Dependencies* or a high *Range Of Types In Subsystems* are most likely not acceptable from a software engineering perspective even if these solutions feature very good performance in the other objectives. Hence, stakeholders might only be interested in a subset of the optimal *Pareto-Front*. However, the definition of thresholds to identify acceptable solutions depends on various factors such as the requirements for the final solution, achievement in the individual objectives, overall performance of the search and available computational resources.

For demonstration purposes it is assumed in the following example that stakeholders do not accept solutions that feature more than 100 architecture violations as it is likely impossible for a stakeholder to be able to estimate the severity of these violations if more architecture violations exist. Additionally, the *Log4J* system features a total of 453 types. It is assumed in this example that stakeholders do not accept solutions that feature a difference between the biggest and smallest subsystem of more than 150 types. Figure 28 highlights a set of promising solutions based on such threshold definitions.

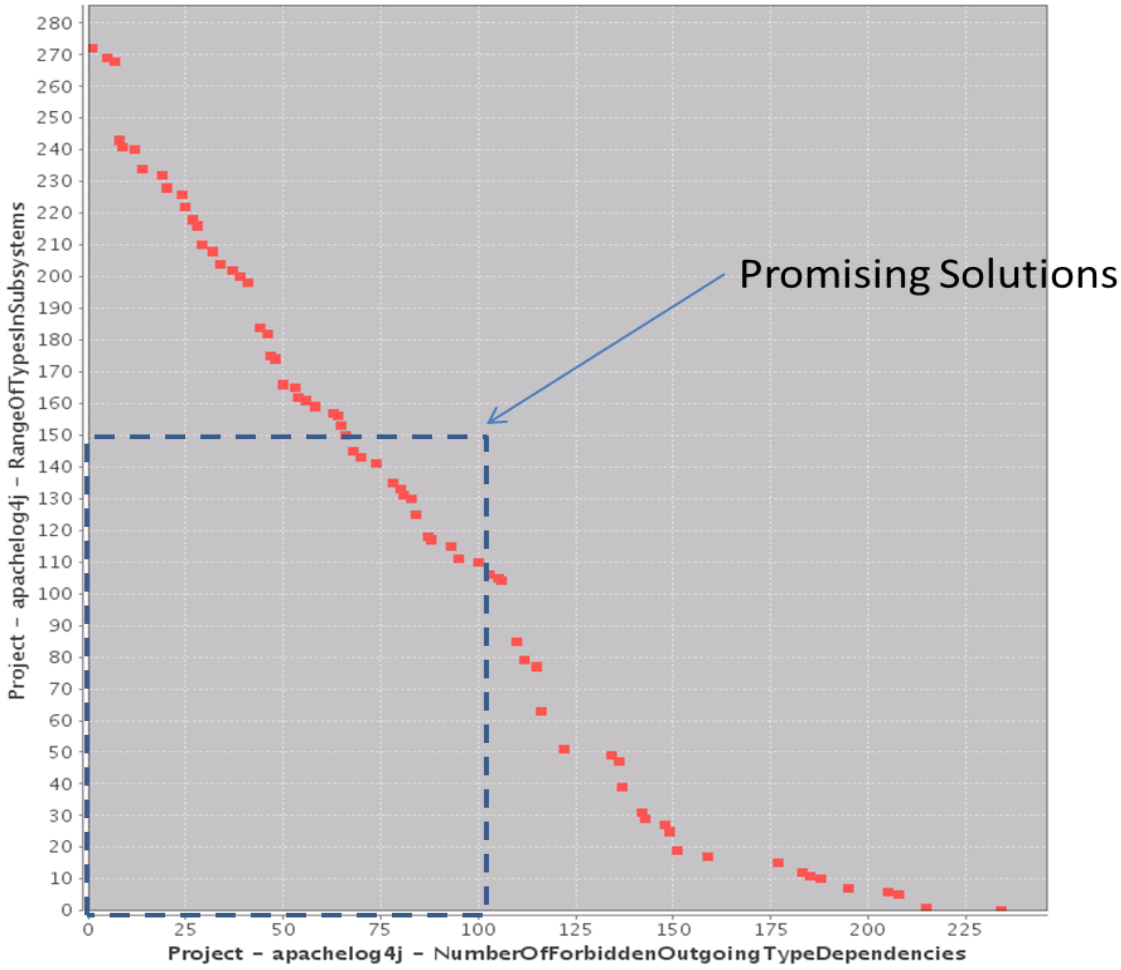


**Figure 28: Promising solutions in a 2D optimisation scenario**

In this example fifteen solutions are within the defined threshold parameters. Hence, the remaining solutions can be labelled as infeasible based on the defined thresholds. The selection of such a set of feasible solutions within a two or three objective optimisation is trivial and does not require any elaborate tool support apart from the implemented visualisation of 2D and 3D sets, zooming (2D and 3D) and rotate (3D) functionality to efficiently identify promising solution sets. However, the identification of promising solution sets based on more than three objectives is not feasible with such techniques. Additionally, this research suggests that there is value in including solutions in the promising region that are not on the optimal *Pareto-Front*. The inclusion of all visited solutions increases the complexity for stakeholders to efficiently

identify promising solutions. The remainder of this section discusses the value of the implemented candidate approaches to constrain solutions to identify promising solutions efficiently in optimisation scenarios with more than three objectives and in solution sets that do not underlie the concepts of optimal *Pareto-Front* dominance.

It has been observed in this research that some metrics feature better convergence than others. For example, *Cohesion*, *Coupling*, *NCCD* and *Distance* usually achieve very good convergence so that (depending on the *MOEA* tuning) each of the solutions of the optimal *Pareto-Front* for those metrics is likely to be acceptable from a stakeholder perspective. However, the achievement of acceptable results in other objectives, such as the minimisation of the number of cycles and the minimisation of the number of architecture violations, seems to be much more difficult. Hence, optimal *Pareto-Front*s might feature solutions with excellent achievement in some objectives but only some solutions with acceptable results in other objectives (compare: Figure 27). This might then only offer stakeholders a limited number of solution candidates that have acceptable performance in the more challenging objectives. Stakeholders therefore might want to accept trade-offs and relax the restrictions of just considering solutions of the optimal *Pareto-Front*s to explore a wider pool of solutions that feature acceptable performance in these more challenging objectives.

Additionally, it has been noted (in section 2.3) that software design metrics are simplified surrogates used to assess the fitness of a solution. For example, it cannot be unequivocally concluded that a solution that features 'better' *Cohesion*, *Coupling* or *NCCD* measurements is actually better from a software engineering perspective. This statement should not be taken as indicating that software design metrics are an infeasible method to determine the quality of a software configuration. Instead, it should rather raise awareness that there is a potential degree of fuzziness that might hinder the exact identification of the most promising solutions.

These problems have been partially addressed by the implementation of multiple optimisation goals, to enable the assessment of software design quality from multiple perspectives and the associated reliance on the concept of *Pareto* dominance that includes any solution that dominates objective space. However, this research also anticipates that there is additional value in considering solutions in the decision making process that are within the acceptable parameters from a development stakeholder perspective, thus enabling stakeholders to select from a wider pool of solutions.

Given the search process, a multitude of different software modularisations are generated and evaluated during a search. Most likely, some of the solutions that are visited during the search feature good convergence and meet the thresholds that are acceptable for the stakeholder. For example, Figure 29 presents the 400,000 solutions that have been visited in the previous optimisation example (comprising two objectives), and the solutions that are considered to be particularly promising, based on the previously discussed thresholds, are highlighted.

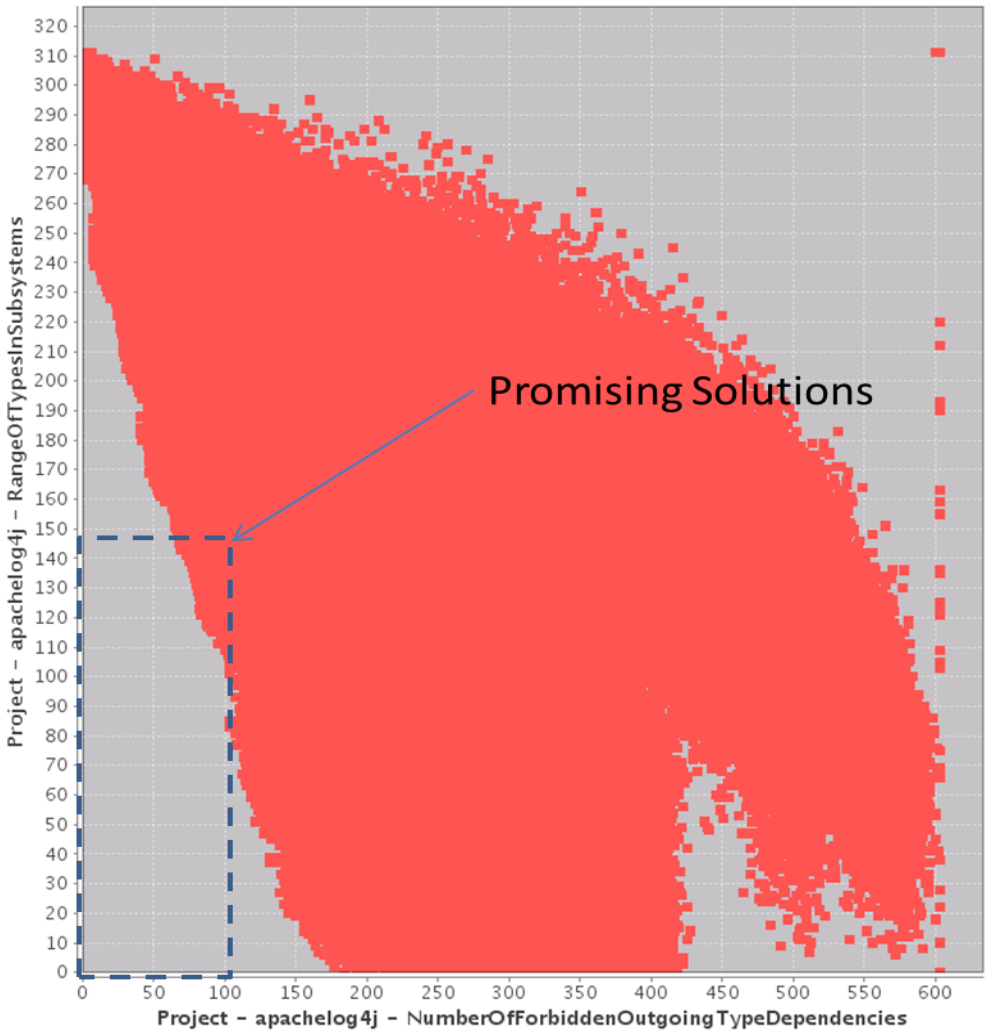


**Figure 29: Visited solutions of the two objective example**

Figure 29 highlights that the presentation of all visited solutions is overwhelming for stakeholders. Correspondingly, stakeholders are most likely unable to efficiently identify a promising solution candidate from the pool of all visited solutions. Hence, tool support is required to confine the set of solutions to a manageable size.

The *Rearchitecturer* prototype therefore enables stakeholders to iteratively constrain solution sets and review individual solutions beyond the limitations of the optimisation settings and the identified optimal *Pareto-Front*. Furthermore, the *Rearchitecturer* component supports stakeholders in discarding solutions from the visualised solution set that feature undesired quality aspects. It is anticipated that such

filter strategies might be a valuable complement to the search configuration in enabling stakeholders to identity a manageable set of highly promising solutions more efficiently.

A component has therefore been implemented to enable stakeholders to constrain the presented solution sets based on the metric measurements of the individual solutions. The *Rearchitecturer* artefact is able to collect a variety of additional solution metrics, and these solution attributes can be utilised to constrain the set of promising solutions. Such a constraint component exists for each of the recorded metrics. Each of these metric constraint components enables the stakeholder to define a range of metric measurements for a particular recorded metric.

Figure 30 depicts two examples of the *Metric Constraint* component. The depicted example confines the visualisation to solutions that feature a *NumberOfForbiddenTypeDependencies* measurement of < 100 and a *RangeOfNumberOfTypesInSubsystems* measurement of < 150.


**Figure 30: Example of the *Rearchitecturer* constraint component**

Each of the bars within the slider component represents a modularisation solution that has been discovered in the search. Multiple solutions can feature the same identical metric value. Hence, multiple solutions can be presented by one bar. The constraining is triggered by moving one of the two (lower and upper) sliders or by entering a value in one or both of the two text fields. As a result, only solutions that feature metric measurements within the desired range of measurements are displayed. Figure 31 depicts the result based on the threshold configurations just discussed.


**Figure 31: Example of a constraint solution set**

The constrained solution set features a total of 2007 solutions. It is evident that it is impractical to review all of the solutions presented in Figure 31. Additionally, the constrained setup only confined the solution set according to the objectives employed in the search. This certainly has only limited value, as simple zooming would have been sufficient to identify such a solution set. However, the *Rearchitecturer* records metrics regardless of whether these are included in the objective setting. Hence, the solution set can be further confined based on attributes that are *not* included in the optimisation. The further review of the constraint component revealed that solutions exist that feature metric values that are not acceptable from a software engeneering perspective. In this example, additional constraints on the *NCCD*, *Coupling* and *Cycle*

metrics have been defined iteratively to reduce the number of solutions. Figure 32 depicts the final constrained configuration that has been identified in this process.

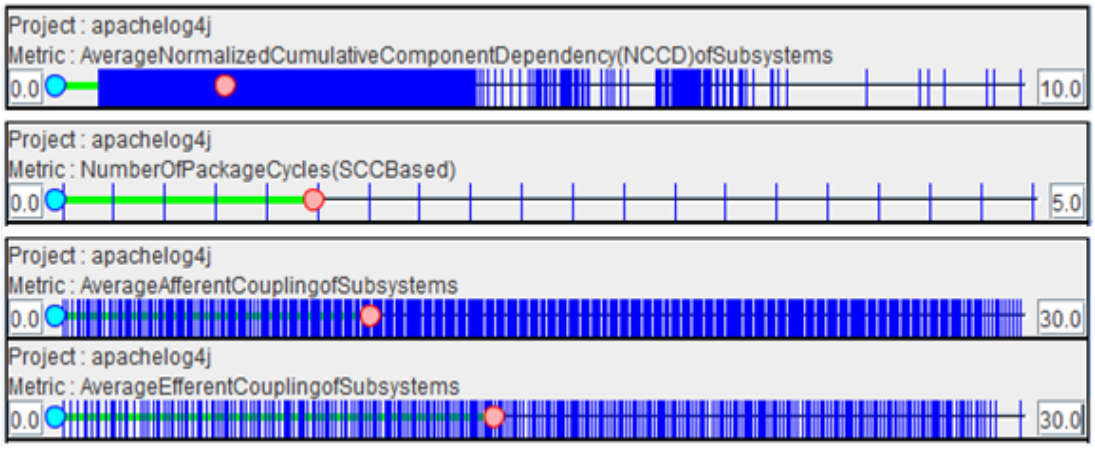


**Figure 32: Constraint configuration of non-objective metrics**

Figure 33 depicts the constrained solution set now based on the constraint configuration presented in Figure 32 and Figure 31.

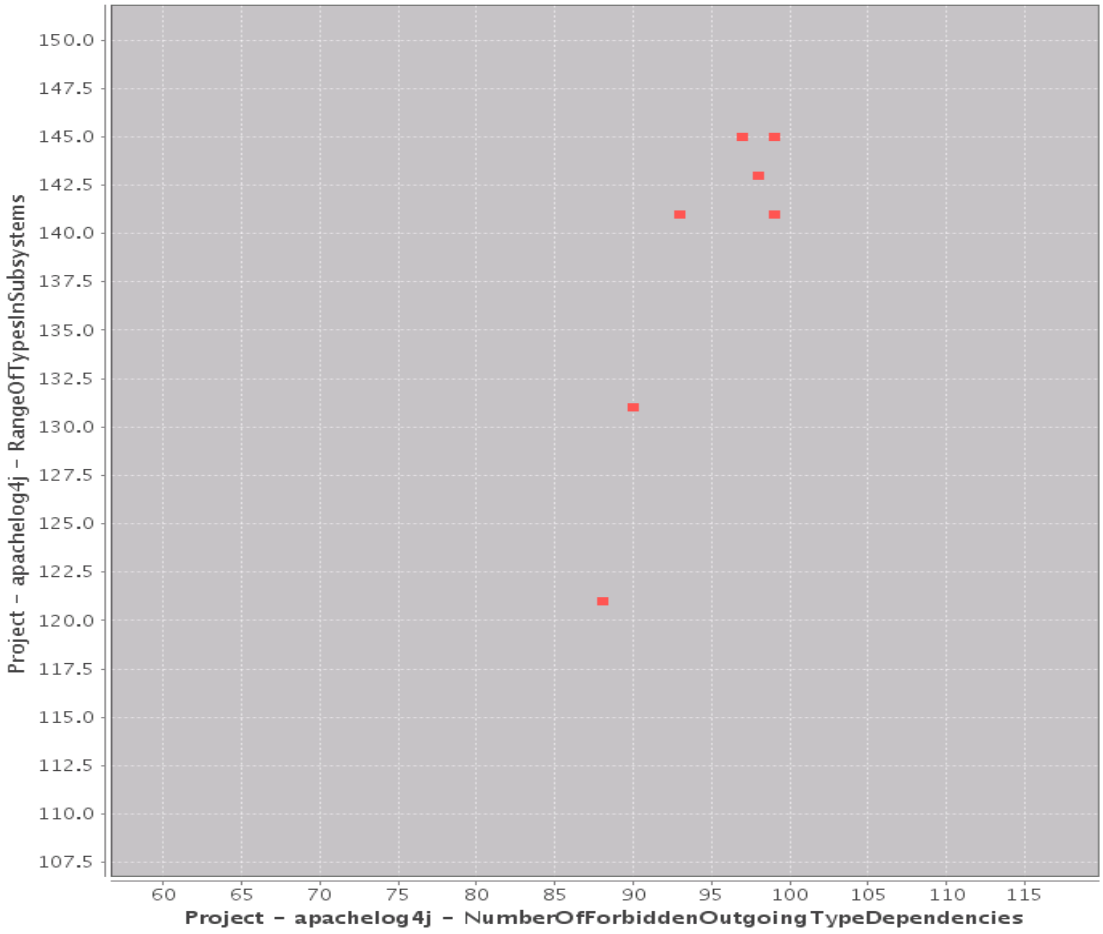


**Figure 33: Solution set based on employed constraint configuration**

Application of the thresholds depicted in Figure 32 reduced the number of solutions from 2007 solutions to a total of 8 solutions. Hence, these 8 solutions from the complete set of the visited solutions (400,000 – compare: Figure 29) meet the thresholds set in Figures 30 and Figure 31. The iterative configuration of thresholds took approximately 30 minutes in this example.

The suggested technique provides an iterative, flexible and user-informed approach to reduce the set of visited solutions incrementally to a manageable set of genuinely promising solutions. The approach enables the interactive exclusion of unacceptable solutions and the acceptance of solutions that meet the requirements of stakeholders. That said, stakeholders can readily relax the constraints if no adequate solutions matching the desired criteria are found. It is anticipated that the application of the

suggested approach is more suitable in the targeted problem domain in comparison to the 'blind' application of the *NDPF* concept that only accepts the best solutions that dominate an area of the objective space. It is assumed in this research that stakeholders are not necessarily seeking the absolute best achievement in an objective dimension but rather prefer solutions that feature good quality in multiple solution aspects.

Another aspect that can be observed in the example just provided is that none of the identified solutions is located on the original optimal *Pareto-Front* (compare: Figure 28). The solutions on the *optimal Pareto-Front* break the quality requirements for solutions defined in the filter configuration (compare Figure 31 and Figure 32), because the filter components constrain attributes that are not included in the original objective setting. Hence, consideration of the complete set of visited solutions enabled the identification of highly promising solutions that originally were not considered based on the concept of optimal *Pareto Dominance*. It can be argued that all of the defined constraints could also be employed as objectives of the search. However, the stakeholder cannot know before the search which solution attributes will feature undesired values. Thus, the up-front definition of the criteria in the objective setting is complex and is most likely never complete. Additionally, the inclusion of more objectives negatively impacts the performance of the search.

The suggested approach is also likely to be helpful if more than three objectives are employed and a visualisation of all the objective dimensions is not feasible. Additionally, the suggested approach can be employed to assure that each of the visualised solutions fulfils a minimum requirement in the non-visualised objectives.

### *4.2.3 Review of Solution Candidates*

The objective goal setting in this research is not understood as the ultimate model of the targeted design. The present research anticipates the *fitness-function* configuration as an abstract model of the desired modularisation design that operates as a point of gravity and consolidates the evolutionary search towards solutions that excel within the defined objective settings. However, other aspects of quality exist that cannot

necessarily be captured with the help of software design metrics, or whose definition within a multi-objective configuration is too complex. Based on the lack of measures that facilitate such support, as highlighted in section 2.5, an objective of the present research is to suggest an approach to support the exploration and review of solution sets to efficiently identify promising solutions.

Stakeholders most likely desire support to review solutions in more detail so as to identify solution attributes or an entire solution that could be applied as an architecture model within the architecture management process. A software component has therefore been developed and integrated into *Rearchitecturer* that enables the visualisation of the partitioning of the physical architecture into a conceptual architecture model. Figure 29 shows an architecture modularisation solution that has been created within the previous search example.

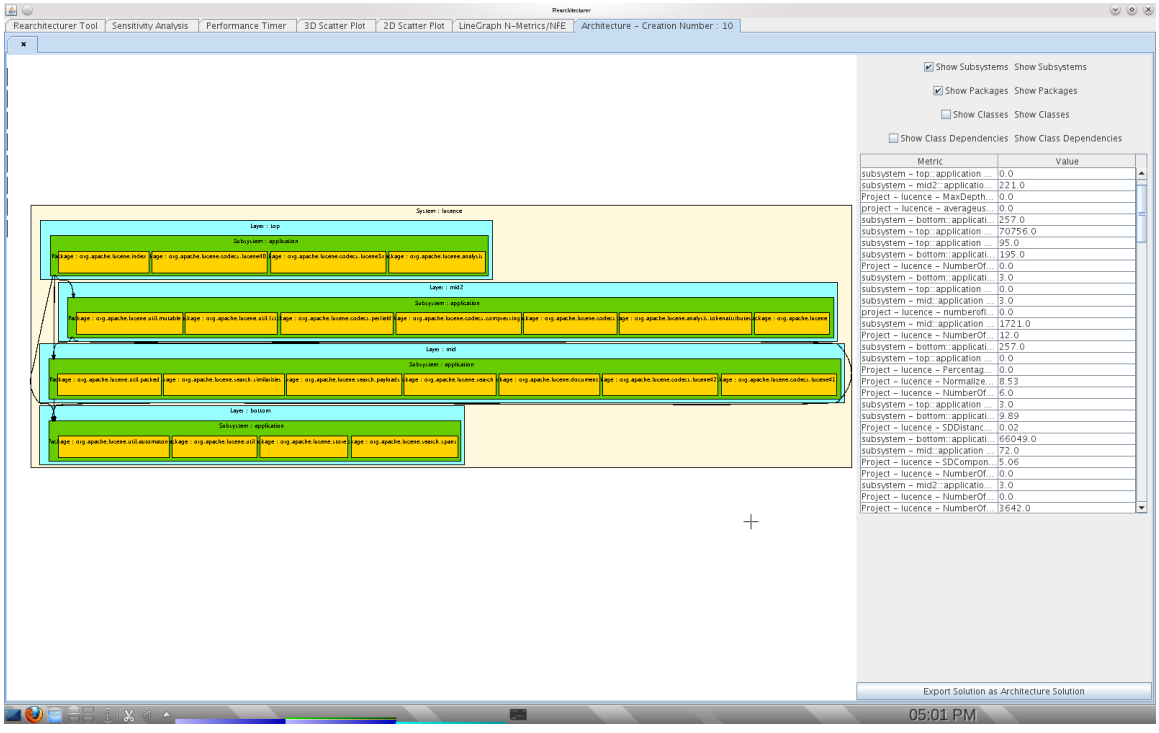


**Figure 34: Example visualisation of an architecture classification solution**

The component supports the visualisation of layers and subsystems and their allowed conceptual dependencies. Furthermore, the classification of packages into subsystems, the classification of compilation units into packages and the physical dependencies between packages and compilation units, based on their physical dependencies on compilation unit level, can be visualised.

Furthermore, the values of all recorded metrics of the displayed modularisation solution are depicted to support the stakeholder in their evaluation of quality aspects of the solution that are beyond the exclusive performance in the objective settings and the capability of the visualisation. Finally, the component offers support to save the conceptual architecture model and classification of the physical architecture artefacts into a XML-file that can be processed by the *Dependometer* architecture management tool. Hence, the solution can be utilised as a development blueprint within the architecture monitoring and management process.

The *ZGRViewer* [38] framework has been utilised to implement the described component to visualise specific architecture modularisations. *ZGRViewer* is a graph visualisation framework specifically designed to handle large graphs. *ZGRViewer* offers a zoomable user interface and easy navigation within the visualized structures (Pietriga, 2005). The *ZGRViewer* framework displays graph models expressed in the DOT language designed by AT&T *GraphViz*[39]. Hence, the DOT executable needs to be installed and accessible in the path of the executing machine to utilise the functionality to visualise specific architecture modularisation solutions within the *Rearchitecturer* artefact. A minor set of source code contributions have been made to the *ZGRViewer* repository. These contributions address features of the cluster-, subgraph- and edge-visualisation and integration of the *ZGRViewer* into SWING components.

[38] http://zvtm.sourceforge.net/zgrviewer.html
[39] http://www.graphviz.org/

## 4.3 *Batch Driven Execution of Search Configurations*

The present research applies multi-objective and randomised algorithms in the problem domain of architecture reconstruction. An objective in the design of the *Rearchitecturer* artefact is to enable stakeholders to configure search parameters in a flexible manner. The current design of the *Rearchitecturer* artefact supports the employment of a vast number of objectives, reconstruction configurations, *MOEA* implementations and *MOEA* tunings. Hence, the design of the *Rearchitecturer* component enables the employment of a variety of architecture reconstruction and *MOEA* configuration scenarios.

The *GUI* component supports the storage of the objective values and decision variables of the visited solutions in a *csv* file. However, the main purpose of the *GUI* component is to support stakeholders in the process of identifying feasible architecture configurations. Hence, the purpose of the *GUI* component is not to provide an interface to collect solution sets for comparative studies.

However, the execution of various configuration settings and multiple reruns of configuration settings is required to enable a representative analysis of the feasibility of the developed approach. For example, the main results that are presented in Chapters 5 and 6 have been collected by employing three different architecture reconstruction scenarios, with six different *MOEA* implementations and five open-source software systems. Hence, the employment of 90 combinations of search configurations has been required for the collection of this main dataset. Additionally, 10 reruns of each configuration are employed to enable robust statistical analysis of the result sets. The number of employed search configurations is even higher if different mutation and crossover tunings are employed in the evaluation (see: section 6.1).

The definition and employment of such a high number of search configurations with the developed *GUI* component of the *Rearchitecturer* component would be tedious. Hence, the *GUI* component is not suitable to conduct a study of the size aspired in this

research. Correspondingly, the employment of a batch-driven execution framework that enables the programmatic execution of search configurations is more suitable to collect the solution sets that are required in the evaluation of this research. Appendix E describes the employment the batch driven execution of search configurations in the Rearchitecturer component.

## 4.4 *Summary*

The present chapter described the design and implementation of the *Rearchitecturer* artefact. It has been emphasised in this chapter that the implemented *Rearchitecturer* artefact suggests a set of novel configuration and solution exploration approaches.

The *Rearchitecturer* artefact enables a flexible approach for the employment of optimisation goals, represented by established software architecture design metrics and reconstruction objectives, to express a desired software architecture design. A variety of state-of-the-art *MOEA* implementations can be employed as search algorithms.

The implemented visualisation approach enables stakeholders to visualise and explore the complex multi-dimensional solution sets. A novel concept that is supported in the *Rearchitecturer* component is to explore the complete set of visited solutions instead of exclusively relying on the presentation of the *optimal Pareto-Front*. The visualisation perspectives of *Rearchitecturer* also enable stakeholders to constrain solution sets based on desired quality aspects to identify particularly promising regions of interest in the solution sets. The designed candidate methods enable the exploration and constraining of solution sets from different software design perspectives, independent of the original optimisation goal configuration. Hence, these suggested candidate techniques can support development stakeholders to gain a deeper insight into the complex inter-dependencies that exist between the different design objectives of the problem domain and to efficiently identify promising solutions independently of the traditional assessment techniques evident in the domain of multi-objective

optimisation. Furthermore, a visualisation approach is presented that enables stakeholders to review solutions individually. Solution candidates can be exported and utilised in the architecture management process to monitor the modularity and architectural conformance of a software system in the ongoing software development process.

It has been demonstrated with an application example that the suggested candidate approaches are feasible methods to consider the design expertise of stakeholders in the solution selection process. The insight gained in this demonstration supports the argument that the exclusive reliance on the concept of non-dominance is not necessarily sufficient to identify the most promising solution candidates. It has rather been demonstrated that promising solutions might exist that exhibit more desirable features but are not included in the optimal *Pareto-Front*. The suggested candidate approaches are not only a contribution to the problem domain of software modularisation and architecture reconstruction. The application of the suggested candidate approaches might also be applicable and valuable in other multi-objective optimisation approaches to identify feasible solution candidates.

# 5 A Multi-Objective Evaluation Framework

Statements on the general feasibility of a *MOEA*-driven architecture reconstruction approach and *MOEA* tunings that lead to solutions exhibiting good performance in the targeted problem domain are required to address the research objectives. In section 3.3.4 and section 3.3.5 relevant aspects for the evaluation of multi-objective approaches and randomised algorithms are discussed. A novel evaluation framework, as an implementation of the discussed multi-objective evaluation concepts, has been developed to facilitate the robust statistical analysis of data sets to address such objectives of the present research.

The application of the evaluation framework is divided into (1.) the employment of a set of search configurations to create data sets of the visited solutions, followed by (2.) the statistical analysis of these data sets. The separation of the collection of the solution sets and the down-stream statistical analysis of those solution sets enables the employment of the implemented evaluation framework in other multi-objective optimisation approaches if these approaches were to adhere with the output format suggested in this research.

The purpose of this chapter is to depict and explain the functionality of the evaluation framework. Output examples of graphs and tables are presented as produced by the multi-objective evaluation framework. However, the data presented in graphs and tables functions as an example to demonstrate the analysis functionality of the evaluation framework, and so the outputs themselves are not necessarily discussed in depth. Chapter 6 applies the evaluation framework on different architecture reconstruction scenarios and discusses the results in more detail.

## 5.1 *Execution of Search Analysis*

The evaluation of this research requires the analysis of multiple search tunings to allow statements on the impact of individual tuning parameters on the search performance. Correspondingly, multiple solution sets are produced in the data

collection process. The analysis framework conducts the statistical analysis of these solution sets. The employment of the analysis framework is described in Appendix F. The analysis results can be classified and then sliced to evaluate performance differences of individual performance tunings.

## 5.2 *Classification of Solution Sets*

The evaluation framework is able to consider the data of multiple solution set directories. The input directories include the *ExperimentConfiguration* instance and the corresponding solution sets. A classification model is built from the solution sets and the attributes of the accompanying *ExperimentConfiguration* instances. More specifically, the classification attributes are the variation operator configuration, the employed *MOEA* implementation, the objective configuration and other meta-information. The meta-information is problem-specific and added into the *ExperimentConfiguration* instances through the execution of call-back functionality during the execution of the search. In this research the meta-information includes the name of the software system that has been used during the employment of the *ExperimentConfiguration* and the design of the applied conceptual architecture. Hence, a model of the *ExperimentConfiguration* and the corresponding solution sets is built. This model is a prerequisite to agglomerate and compare the performance of configuration attributes in different configuration scenarios.

## 5.3 *Creation of Non-Dominated Pareto-Fronts (NDPF)*

A *Non-Dominated Pareto-Front (NDPF)* is created for each seed of the solution set. Additionally, a true or best *NDPF* is required for the calculation of performance metrics that measure solution convergence (compare section: 3.3.4). An approximation of a true *NDPF* is created by merging the individual *NDPFs* of the individual seeds of the solution sets.

A challenge in the evaluation of *MOEAs* is that the merging of *NDPFs* is not necessarily meaningful in all evaluation scenarios. For example, the merging of *NDPFs* of

different software systems might not be seen as desirable if the performance of *MOEAs* across different software systems is evaluated based on the analysis of the achievement in multi-objective performance indicators. The reason for this is that software systems feature different levels of complexity (e.g. different numbers of artefacts and dependencies). As a result the optimisation in the different objectives (e.g. minimisation of architecture violations or cycles) is generally simpler in less complex software systems. Hence, it is likely that the solutions of a *NDPF* of a less complex system will feature better convergence towards an optimal solution. Correspondingly, the solutions of the *NDPF* of the less complex system will dominate the *NDPF* of the more complex software system if *NDPFs* of different software systems are merged. Hence, there is a chance that the more complex system is excluded from the performance analysis. However, stakeholders are interested in identifying *MOEA* implementations and tunings that perform well across different kinds of system sizes. Hence, it is suggested in the present research that *NDPFs* should be contained in separate *NDPF* pools to overcome this problem. A best *NDPF* is calculated for each pool of *NDPFs*.

The evaluation framework therefore implements the suggested approach that enables the separate handling of *NDPFs*. Performance metrics are calculated for each *NDPF* in relation to the best *NDPF* of the pool (see section 5.4). The calculated performance indicators of the individual *NDPF* criteria pools are normalised to allow a comparison and consolidation of result sets across *NDPF* criteria pools.

The *software system* attribute is utilised as a *NDPF* pooling criterion in the evaluation of this research that analyses the achievement of convergence-based performance indicators across multiple systems.

## 5.4 *Calculation of Performance Metrics*

The evaluation framework enables the statistical analyses of the development of individual objective values and unary *Pareto-Front* performance indicators. The reporting of objective values enables statements to be made on solution achievement

from a problem domain perspective. However, the reporting of individual objective achievements has the disadvantage that only statements on performance in one objective dimension at a time are possible.

The advantage of the application of unary performance indicators is that a single value is reported that expresses the goodness of the optimal *Pareto-Front*. This includes the progress in all objective dimensions. However, the performance indicators are domain independent. Consequently, the performance indicators only give a relative assessment of the goodness of an optimisation run. Hence, the consideration of *Pareto-Front* performance metrics *in combination with* the assessment of objective achievement is useful to assess the performance of optimisation runs. The objective achievement is conducted on the basis of the objective achievement within the generational populations or the objective achievement of the optimal *Pareto-Front*. Hence, all solutions of a generation or the optimal *Pareto-Front* are consolidated.

The implemented evaluation approach calculates the performance metrics *Spacing*, *Contribution*, *Generational Distance*, *Additive Epsilon Indicator* and *Hypervolume*. The convergence performance metrics are calculated for each seed by using the calculated best *NDPFs* of the corresponding *NDPF* classification criteria set (software system in the *architecture reconstruction* problem).

The performance indicators and the objective achievement are not calculated only once at the end of the search as is most common in prior research. The present research anticipates that the reporting of performance metrics and objective progress at multiple time points is a valuable evaluation approach to enable statements to be made regarding the development of the search. Hence, the performance metrics are calculated at multiple iteration points of the search, based on the defined frequency.

The calculated performance indicators of the individual *NDPF* criteria pools are normalised to enable the comparison and consolidation of result sets across *NDPF*

criteria pools. The calculation of the performance indicators is executed in parallel, based on the defined number of cores in the *AnalysisConfiguration* instance.

## 5.5 *Slicing of Search Configurations (Seeds)*

Both Sayyad and Ammar (2013) and Arcuri and Briand (2011) state that *SBSE* research generally claims contributions based on the employment of a single search configuration. A typical use case is the application of a single *MOEA* implementation with one set of variation operators in a single problem scenario (Arcuri & Briand, 2011). However, the applied *MOEA*, the variation operator tuning and the problem scenario can potentially have a noticeable effect on the performance of the approach. The present research anticipates that a more comprehensive evaluation approach is needed to support thorough conclusions on the feasibility and performance of a search based problem solution. This research suggests that the agglomeration and classification of solution sets based on configuration attributes provide such an approach. The agglomeration of search configuration attributes is useful to gather confidence in the general feasibility of an approach. For example, the agglomeration of the *software system* property is a useful measure to enable statements to be made on the general performance of the approach and configuration across different systems. However, to gain insights into the impact on the performance of individual configuration attributes a separation of solution sets based on search configuration attributes is also useful.

For instance, the agglomeration of the analysed systems coupled with the separation of the employed *MOEA* implementations is a useful means to analyse the performance of individual *MOEAs* across multiple systems. Another evaluation scenario is, for example, to assess the impact of variation operator settings across different software systems, *MOEA* configuration and architecture reconstruction settings.

The evaluation framework creates a result file for any combination of agglomeration and classification of the search criteria. The result files feature the mean of the

performance metrics of the agglomerated solution sets at the defined frequency intervals up to the defined termination criteria.

The remainder of this section depicts an application example of the evaluation framework in which a set of *MOEA* implementations across different systems and variation operator settings is analysed. The software system attribute, variation operator setting and conceptual architecture configuration are agglomerated and a classification based on the *MOEA* implementation is conducted.

The solution sets have been collected in a set of experiments in which 8 objectives are employed (see section 2.3 for a detailed description of the employed objectives) to reconstruct the architecture classification of a set of five software systems (*Lucene, Apache Ant, Apache Math, Apache Log4j, Rearchitecturer*). One reconstruction method is applied in which compilation units are assigned to the existing packages and the packages are assigned into three different conceptual architecture models that feature four layers. Sections 6.1 and 6.2 describe the conceptual architecture design and experiment setup in more detail. Six algorithms (*MOEA*, *AbYSS*, *NSGAII*, *OMOPSO*, *GDE3*, *Random*) are executed with each of these configurations. This results in a total of 90 different search configurations (6 MOEA implementations x 5 software system x 3 reconstruction scenarios). 10 seeds are collected for each evaluation scenario. Hence, this evaluation scenario created a total of 900 solution sets (90 search configurations x 10 seeds). The termination criterion is set to 50,000 solution evaluations. Hence, each solution set features a total of 50,000 visited solutions. The frequency is set to 10,000, meaning that a snapshot of the performance indicators is calculated every 10,000 solutions. Such a low frequency has been selected to enable the presentation of snapshots across the complete search due to space restrictions in the presented tables of results. In the later evaluation of this research a higher frequency of 100 is used to enable a more accurate recording of the performance development of the search.

In the first agglomeration example the performance of the *MOEA* implementations is compared. Correspondingly, the evaluation framework separates the solution sets (900) into six algorithm groups. Each group features a total of 150 (900 /6 algorithms) solution sets. Each of these groups contains solution sets that have been created by employing search configurations that feature the five different software systems.

Table 3 depicts the mean of the *Hypervolume* performance metric and gives information on the agglomeration of the individual search configuration attributes.

**Table 3:** ***Development of Hypervolume – Slicing by MOEA implementation***

| Algorithm | System | Mutation Rate | NrAgglSeeds | 10000 | 20000 | 30000 | 40000 | 50000 |
|---|---|---|---|---|---|---|---|---|
| AbYSS | * | * | 150 | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 |
| GDE3 | * | * | 150 | 0.02 | 0.04 | 0.05 | 0.06 | 0.07 |
| *MOEA*D | * | * | 150 | 0.02 | 0.03 | 0.05 | 0.06 | 0.07 |
| NSGAII | * | * | 150 | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 |
| OMOPSO | * | * | 150 | 0.32 | 0.36 | 0.38 | 0.40 | 0.41 |
| Random | * | * | 150 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

*agglomerated parameter

Information on other classification attributes such as information on the population size, other variation operator information and criteria has been removed from the original evaluation framework output to enable clearer presentation in this thesis. Nevertheless, the analysed data in this example does not feature differences in these attributes. Figure 35 graphically depicts the development of the mean *Hypervolume* values as presented in Table 3.

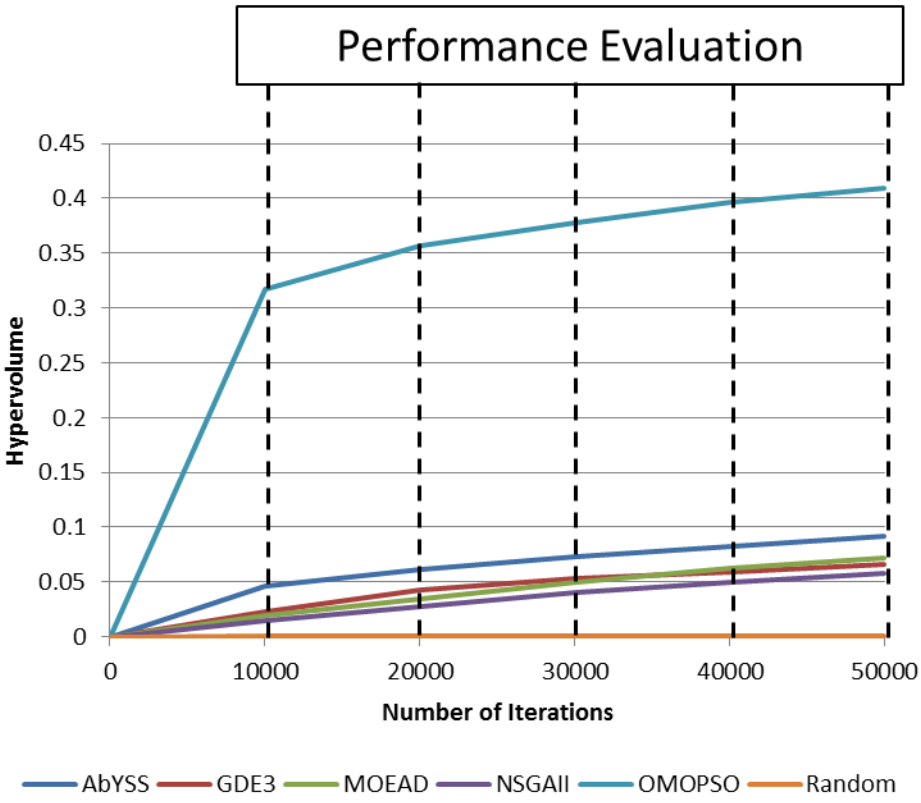


**Figure 35: Development of *Hypervolume* – Slicing by MOEA implementation**

The information presented in Table 3 and Figure 35 is useful to assess the mean performance of the agglomerated seeds in the individual *MOEA* implementations. The *Hypervolume* measures the dominated space of an optimal Pareto-Front in relation to a best or reference *Pareto-Front* (compare section 3.3.4). Hence, a higher *Hypervolume* measure indicates better performance of a slice.

The evaluation framework can also be used to analyse the impact of other configuration aspects. Five different probabilities of *Polynomial Mutation* rates are compared in the following example. See section 6.1 for details on the applied *Polynomial Mutation* strategy. A data set is collected that features the previous configuration and five different *Polynomial Mutation* rate settings are applied. Obviously, *Random* search is not impacted by different mutation rate tunings and so is excluded from this analysis. A total of 375 search configurations are employed in this

example (5 algorithms x 5 software systems x 3 reconstruction scenarios x 5 mutation rate settings). Each search configuration is executed 10 times. Hence, a total of 3,750 seeds are executed. The collected solution sets are classified into five groups (pm.rate 0.0, pm.rate 0.1, pm.rate 0.3, pm.rate 0.5, pm.rate 0.7) as the mutation rate is used as the agglomeration criterion. Each group features a total of 750 seeds. Table 4 presents the *Hypervolume* results in table format as created by the evaluation framework.

**Table 4: *Development of Hypervolume – Slicing by Mutation Rate***

| Algorithm | System | Mutation Rate | Seeds | 10000 | 20000 | 30000 | 40000 | 50000 |
|---|---|---|---|---|---|---|---|---|
| * | * | 0.0 | 750 | 0.066 | 0.083 | 0.095 | 0.104 | 0.110 |
| * | * | 0.1 | 750 | 0.068 | 0.084 | 0.095 | 0.103 | 0.112 |
| * | * | 0.3 | 750 | 0.067 | 0.083 | 0.095 | 0.106 | 0.113 |
| * | * | 0.5 | 750 | 0.074 | 0.091 | 0.103 | 0.111 | 0.118 |
| * | * | 0.7 | 750 | 0.068 | 0.085 | 0.096 | 0.104 | 0.111 |

*agglomerated parameter

Figure 36 depicts the corresponding graph visualisation of the results.

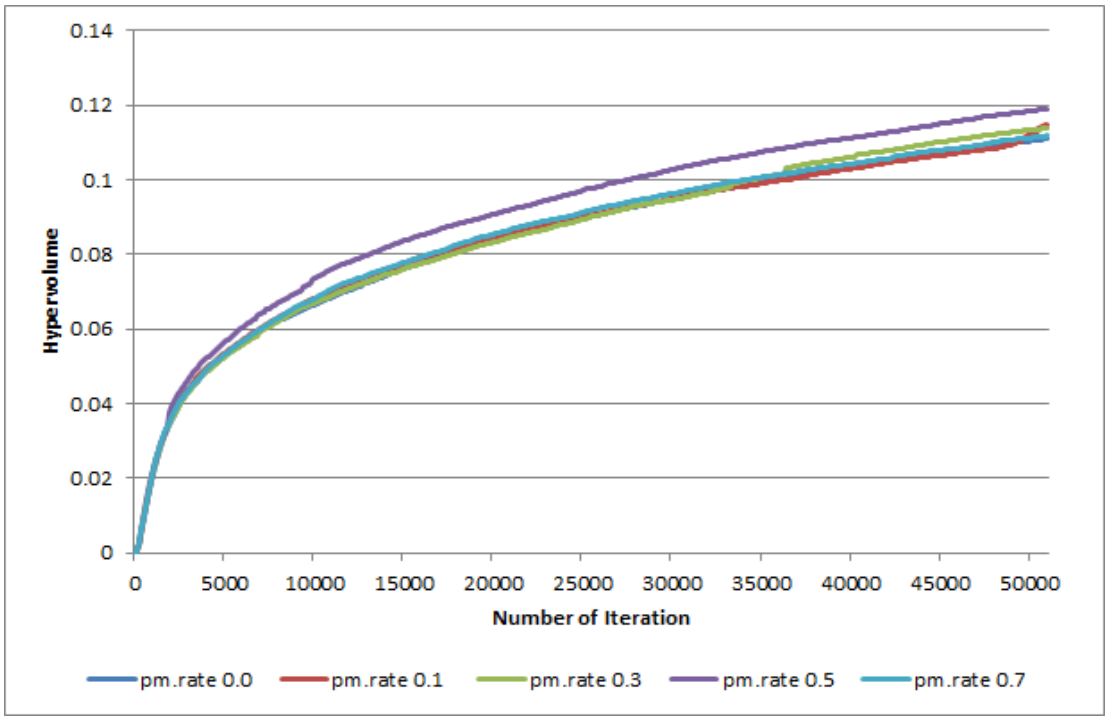


**Figure 36: Development of *Hypervolume* – Slicing by Mutation Rate**

This example demonstrates the slicing of the same data sets to assess the impact of different mutation rate tunings across multiple *MOEA* implementations. Hence, the illustrated examples show the feasibility of the evaluation framework to slice data sets based on different criteria to investigate the impact of different tuning parameters (*MOEA* implementation and mutation rate). Additionally, the agglomeration of datasets with different parameter tunings of non-investigated criteria enables more general statements to be made on the impact of a specific tuning. The interface of the evaluation framework supports the user in agglomerating any combination of search configuration parameters. If no slicing criteria are defined each search configuration is depicted separately and no agglomeration of solution sets is conducted. The slicing is conducted for each of the six implemented performance metrics (*Spacing*, *Contribution*, *Generational Distance*, *Inverted Generational Distance, Additive Epsilon Indicator*, and *Hypervolume*).

## 5.6 *Statistical Comparison of Performance*

In the previous slicing example the mean development of a performance indicator from a population of seeds is presented (compare: Table 3 and Table 4). The mean of a performance indicator is probably the most common descriptive statistic in a performance comparison scenario. However, the interface of the evaluation framework allows the output of multiple descriptive statistic measures. The supported descriptive statistic measures are the mean, median, minimum, maximum, kurtosis, skewness and standard deviation.

As discussed previously each *MOEA* implementation applied methods of random variation. Inductive statistics need to be applied to determine if a population features significant performance differences. Sufficiently large populations of individuals are therefore needed to support the conduct of statistical analysis. Populations of performance metric measures are created based on a defined range of performance snapshots. Descriptive statistics (mean, median, minimum, maximum, kurtosis, skewness, standard deviation and Kolmogorov-Smirnov p-value) are calculated for the

generated populations. In this example, the *Hypervolume* performance of the performance snapshot at iteration 50,000 is presented. This analysis example considers the same dataset as used in section 5.5. A detailed description on the configuration of this dataset can be found in section 6.2. The dataset is sliced by the applied *MOEA* implementation. Hence, each slice features a population of 150 individuals. The following equation gives a more formal description of the calculation of the population size:

$$\frac{(6 \text{ algorithms x } 5 \text{ software systems x } 3 \text{ reconstruction scenarios x } 1 \text{ variation operator setting x } 10 \text{ seeds})}{6 \; slicing \; groups \; (MOEA \; implementations)}.$$

Table 5 presents the corresponding output gathered by the evaluation framework.

**Table 5: Descriptive Statistics - *Hypervolume* (Iteration 50,000)**

| Algorithm | Mean | Median | Max | Min | SD | Kurtosis | Skewness | KS | N |
|---|---|---|---|---|---|---|---|---|---|
| **AbYSS** | 0.0913 | 0.0602 | 0.3818 | 0.0074 | 0.0843 | 1.0938 | 1.4307 | 0 | 150 |
| **GDE3** | 0.0657 | 0 | 0.2325 | 0 | 0.0642 | -0.6066 | 0.671 | 0 | 150 |
| ***MOEA*D** | 0.0725 | 0 | 0.3542 | 0 | 0.0912 | 0.7097 | 1.3286 | 0 | 150 |
| **NSGAII** | 0.0575 | 0 | 0.2667 | 0 | 0.0602 | 0.1944 | 0.8645 | 0 | 150 |
| **OMOPSO** | 0.4098 | 0.3326 | 1 | 0.0717 | 0.224 | -0.4711 | 0.3439 | 0 | 150 |
| **Random** | 0.0011 | 0 | 0.0085 | 0 | 0.0021 | 2.1354 | 1.8604 | 0 | 150 |

The comparison of the reported descriptive statistics facilitates the general performance assessment of the individual slice populations. However, these descriptive statistics present only an aggregation of a set of snapshots from the search development presented in Table 3. The presentation of the development of descriptive statistics by itself is not sufficient to determine the statistical difference of the populations. The evaluation framework therefore supports the automated statistical significance testing of the population slices at certain points of the search.

The evaluation framework uses the p-value of the *Kolmogorov-Smirnov Normality* to determine if the normality condition is fulfilled within the individual populations. The p-value of the *Kolmogorov-Smirnov* test determines the appropriate method to test

statistical significance. A one-way *ANOVA* is applied if the normality condition is fulfilled in all populations, whereas the non-parametric *Kruskal-Wallis* test is applied if the normality condition is not fulfilled in at least one of the populations. The one-way *ANOVA* or *Kruskal-Wallis test* determines if any significant differences between the data sets exist. Depending on the fulfilment of the normality condition the *student t-test* or *Mann-Whitney* test is applied to calculate the significance of differences for the individual population pairs. In general, the normality condition is not fulfilled in the datasets that have been analysed in the evaluation of this research (compare: Table 5). Correspondingly, the framework applies the *Kruskal-Wallis* test and *Mann-Whitney* tests in the down-stream post-hoc analysis. The *Kruskal-Wallis p value* reports that a significant difference ( < 0.001) between the populations of the discussed example data sets exist.

In the post-hoc analysis a significance test is conducted for each pair of populations to determine the statistical differences between the individual population pairs. Hence, a high number of significance tests are conducted depending on the number of populations. If many significant comparisons are conducted the significance level needs to be reduced to limit the risk of false positive significance outcomes.

The evaluation framework therefore applies the *Bonferroni* correction to calculate a new significance level. The calculation of the *Bonferroni* correction is described in section 3.3.5. In the discussed example the new significance level is 0.0033. Table 6 presents the output of p-values of the *Mann-Whitney* tests.

**Table 6: *Hypervolume* Mann-Whitney – p-value (Iteration 50,000)**

| Algorithm | AbYSS | GDE3 | *MOEA*D | NSGAII | OMOPSO | Random |
|---|---|---|---|---|---|---|
| **AbYSS** | 1.0000 | **0.0015** | **0.0000** | **0.0000** | **0.0000** | **0.0000** |
| **GDE3** | **0.0015** | 1.0000 | 0.6758 | 0.0932 | **0.0000** | **0.0000** |
| ***MOEA*D** | **0.0000** | 0.6758 | 1.0000 | 0.2399 | **0.0000** | **0.0000** |
| **NSGAII** | **0.0000** | 0.0932 | 0.2399 | 1.0000 | **0.0000** | **0.0000** |
| **OMOPSO** | **0.0000** | **0.0000** | **0.0000** | **0.0000** | 1.0000 | **0.0000** |
| **Random** | **0.0000** | **0.0000** | **0.0000** | **0.0000** | **0.0000** | 1.0000 |

The significance table shows that there is a significant difference between *Random*, *OMOPSO, AbYSS* and any other algorithm. No sufficient statistical differences can be reported between *GDE3*, *MOEAD* and *NSGAII*. However, these statistical results do not permit any conclusions to be drawn on the actual difference of the individual algorithms. Correspondingly, the actual difference of the mean and/or median in combination with the statistical significance needs to be assessed to make a conclusion on the actual difference of the sample size. Additionally, the mean/median might not be sufficient to make conclusions on the reliability of the algorithm performance. Hence, the *SD* is another indicator that might need to be considered. The comparison of such multiple outputs is most likely necessary to enable a justified conclusion on the performance of individual populations.

The evaluation framework calculates the *Cohen's d* effect size to assess the magnitude of any difference. Cohen (2013) defines the *Cohen's d* effect size as the difference between two means divided by the standard deviation. The evaluation framework calculates *Cohen's d* as follows:

$$Cohen's\ d = \frac{Mean(population1) - Mean\ (population2)}{\frac{SD(population1) + SD\ (population2)}{2}}$$

A *Cohen's d* measure of 0.2 equates to a small effect, 0.5 equates to a medium effect, and greater than 0.8 equates to a large effect. Table 7 depicts the *Cohen's d* effect size measures of the discussed example.

**Table 7: *Hypervolume* – Effect-Size *Cohen's d* (Iteration 50,000)**

| Algorithm | AbYSS | GDE3 | *MOEA*D | NSGAII | OMOPSO | Random |
|---|---|---|---|---|---|---|
| **AbYSS** | 0.0000 | -0.3444 | -0.2143 | -0.4670 | 2.0661 | -2.0859 |
| **GDE3** | 0.3444 | 0.0000 | 0.0872 | -0.1310 | 2.3875 | -1.9457 |
| ***MOEA*D** | 0.2143 | -0.0872 | 0.0000 | -0.1972 | 2.1402 | -1.5285 |
| **NSGAII** | 0.4670 | 0.1310 | 0.1972 | 0.0000 | 2.4792 | -1.8115 |
| **OMOPSO** | -2.0661 | -2.3875 | -2.1402 | -2.4792 | 0.0000 | -3.6144 |
| **Random** | 2.0859 | 1.9457 | 1.5285 | 1.8115 | 3.6144 | 0.0000 |

Positive values indicate that the population named in the horizontal heading features higher values whereas negative values show that the population of the vertical description features higher values. Positive effect size measures do not necessarily translate to better performance. For example, the *Additive Epsilon Indicator*, *Generational Distance* and *Inverted Generational Distance* indicate better performance if smaller values are reported. Hence, for these metrics, negative values indicate better performance. In contrast, positive *Hypervolume* measures indicate better performance. Table 7 indicates that *AbYSS*, *GDE3*, *MOEAD* and *NSGAII* feature a large effect size in comparison to *Random*. Additionally, *AbYSS* features a medium effect size in comparison to *GDE3*, *MOEAD* and *NSGAII*. Finally, the large effect sizes in the *OMOPSO* column show that *OMOPSO* distinctly outperforms all other algorithms.

The discussed example has only been demonstrated based on the *Hypervolume* performance metric: This example of the application of the evaluation framework compared the *Hypervolume* performance of a set of algorithms across multiple systems and mutation rate settings. Nevertheless, the evaluation framework supports the same statistical analysis for other performance metrics (e.g. *Generational Distance*, *Inverted Generational Distance*, *Additive Epsilon Indicator*, *Spacing*, and *Contribution*) and also for any recorded objective or non-objective solution attribute. Hence, the evaluation framework provides a comprehensive and flexible approach enabling users to assess and compare the performance of slices of datasets based on the statistical analysis of multiple performance indicators.

## 5.7 *Performance Development Analysis*

The previous examples demonstrate the statistical analysis of dataset slices based on the final snapshot of the search. This is the approach that is typically used to evaluate the performance of optimisation approaches. However, through the evaluation of this research it has been observed that scenarios exist in which different configuration settings perform differently through the course of the search. For example, Figure 37

presents the mean development of the *Pareto-Front* in the *Number of Package Cycles* objective.

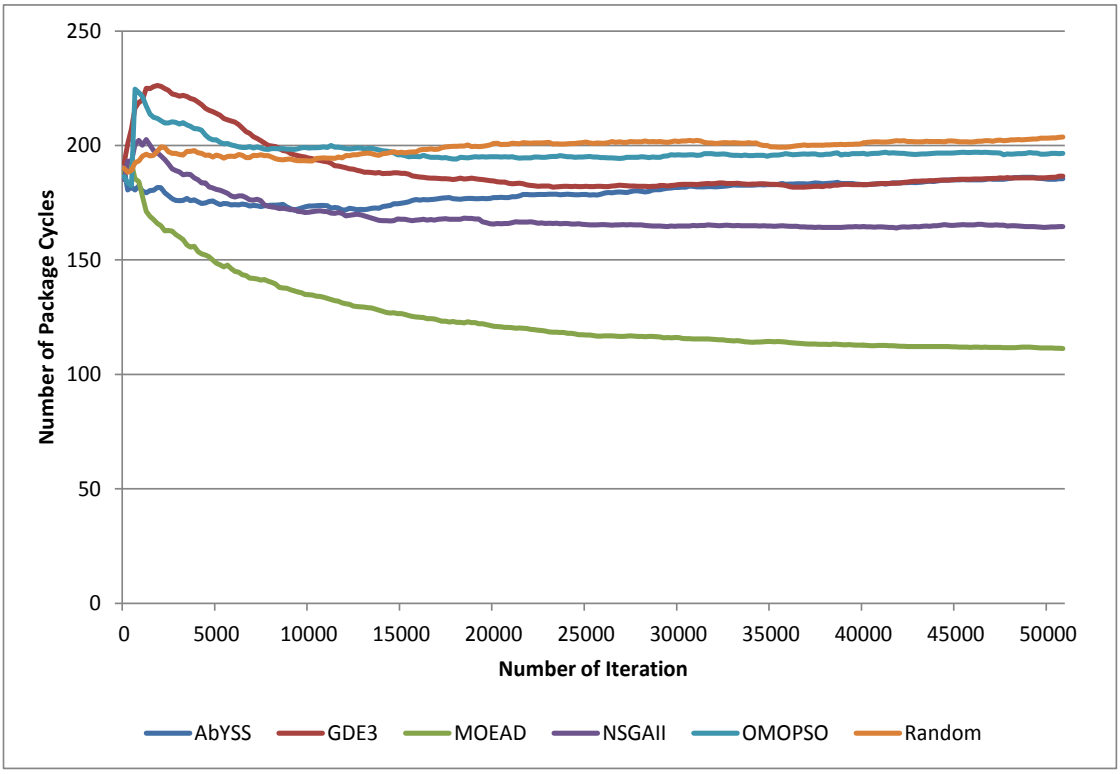


**Figure 37: Mean of optimal *Pareto-Front* in *Number of Package Cycles Objective***

The example depicts that the time point of the analysis impacts the outcome of a performance evaluation. For example, the mean performance of the *GDE3* (M = 186.11) algorithm is very similar to the performance of *AbYSS* (M = 185.29) at iteration 50,000. However, there is a noticeable performance difference through the course of the search, in that *AbYSS* features a better average performance during the development of the search. Such a phenomenon has been observed in the development of performance indicators as well as in the development of objective achievement measures as presented in the example above.

It has therefore been determined in the developed evaluation framework that the consideration of a *range* of snapshots in the statistical analysis might be a valuable extension in the evaluation of multi-objective search results to enable the user consider

differences in performance through the search process as well as the rapidness of convergence.

Multiple analysis snapshots are taken based on the evaluation configuration. Additionally, the configuration enables the definition of a range in which the performance snapshots are consolidated. Populations of performance snapshot measures are therefore created based on the defined range. Descriptive statistics, measures of statistical difference and effect size measures are calculated for the combined populations as presented in section 5.6. Table 8 presents the descriptive statistics at the end of the search (analysis snapshot at iteration 50,000). Table 9 presents the range agglomeration of the entire search (iteration 0 -50,000). A performance snapshot interval of 100 is applied in this example. Hence, a performance snapshot is calculated every 100 iterations. The population is created based on the data collected in these performance snapshots.

**Table 8: Descriptive Statistics - Number of Package Cycles (Iteration 50,000)**

| Algorithm | Mean | Median | Max | Min | SD | KS Normality | N |
|---|---|---|---|---|---|---|---|
| **AbYSS** | 185.30 | 89.23 | 620.00 | 15.00 | 186.79 | 0.00 | 3607.00 |
| **GDE3** | 186.12 | 103.30 | 620.00 | 19.00 | 180.21 | 0.00 | 4880.00 |
| **MOEAD** | 111.43 | 62.15 | 590.00 | 18.00 | 129.63 | 0.00 | 4635.00 |
| **NSGAII** | 164.50 | 87.04 | 630.00 | 19.00 | 171.70 | 0.00 | 4356.00 |
| **OMOPSO** | 196.50 | 90.68 | 600.00 | 2.00 | 188.13 | 0.00 | 3898.00 |
| **Random** | 203.29 | 88.21 | 600.00 | 19.00 | 212.98 | 0.00 | 1662.00 |

**Table 9: Descriptive Statistics - Number of Package Cycles (Iteration 0-50,000)**

| Algorithm | Mean | Median | Max | Min | SD | KS Normality | N |
|---|---|---|---|---|---|---|---|
| **AbYSS** | 180.74 | 85.95 | 620.00 | 15.00 | 187.17 | 0.00 | 611253.00 |
| **GDE3** | 186.15 | 102.64 | 620.00 | 19.00 | 181.32 | 0.00 | 854369.00 |
| **MOEAD** | 119.40 | 63.82 | 600.00 | 18.00 | 139.38 | 0.00 | 813862.00 |
| **NSGAII** | 166.78 | 87.29 | 630.00 | 19.00 | 174.47 | 0.00 | 754854.00 |
| **OMOPSO** | 196.62 | 91.93 | 600.00 | 2.00 | 189.04 | 0.00 | 692638.00 |
| **Random** | 199.93 | 85.76 | 610.00 | 19.00 | 213.43 | 0.00 | 325170.00 |

It has been highlighted in the discussion of the results of Figure 37 that *AbYSS* and *GDE3* feature a similar mean performance at iteration 50,000 but that their performance through the search features some differences. This is supported by the results of the performance snapshot taken after 50,000 iterations, in which *AbYSS* and *GDE3* have an almost identical mean value (compare: Table 8). Consideration of the complete set of multiple performance snapshots, however, confirms a lower mean value of the *AbYSS* population (M=180.74) in comparison to the *GDE3* population (M=186.15) (compare: Table 9).

As discussed in the previous section significance analysis is conducted by the evaluation framework to assess statistical differences between the individual populations. The *Kruskal-Wallis* test showed that statistical differences between the individual populations in both performance analyses exist. Additionally, the conducted pair-wise significance comparisons showed statistical significance for all pairs in the two analyses examples. However, as discussed in section 3.3.5, the high number of individuals in the analysed populations (compare: Table 8 and Table 9) increases the likelihood of the rejection of the null hypothesis even with marginal differences in effect sizes. The presentation of the results of the statistical difference analysis are omitted due to their limited informative value. Nevertheless, the result sets can be downloaded from the project webpage[40].

The effect size therefore needs to be considered to enable statements to be made on the magnitude of the differences between the populations. Table 10 and Table 11 present the *Cohen's d* measures of the mean advancement of the two analyses.

[40] http://code.google.com/p/rearchitecturer/wiki/evaluationResults

**Table 10: Effect-Size *Cohen's d* - Number of Package Cycles (Iteration 50,000)**

| Algorithm | AbYSS | GDE3 | *MOEA*D | NSGAII | OMOPSO | Random |
|---|---|---|---|---|---|---|
| **AbYSS** | 0.000 | 0.004 | -0.467 | -0.116 | 0.060 | 0.090 |
| **GDE3** | -0.004 | 0.000 | -0.482 | -0.123 | 0.056 | 0.087 |
| ***MOEA*D** | 0.467 | 0.482 | 0.000 | 0.352 | 0.535 | 0.536 |
| **NSGAII** | 0.116 | 0.123 | -0.352 | 0.000 | 0.178 | 0.202 |
| **OMOPSO** | -0.060 | -0.056 | -0.535 | -0.178 | 0.000 | 0.034 |
| **Random** | -0.090 | -0.087 | -0.536 | -0.202 | -0.034 | 0.000 |

**Table 11: Effect-Size *Cohen's d* - Number of Package Cycles (Iteration 0-50,000)**

| Algorithm | AbYSS | GDE3 | *MOEA*D | NSGAII | OMOPSO | Random |
|---|---|---|---|---|---|---|
| **AbYSS** | 0.000 | 0.029 | -0.376 | -0.077 | 0.084 | 0.096 |
| **GDE3** | -0.029 | 0.000 | -0.416 | -0.109 | 0.057 | 0.070 |
| ***MOEA*D** | 0.376 | 0.416 | 0.000 | 0.302 | 0.470 | 0.457 |
| **NSGAII** | 0.077 | 0.109 | -0.302 | 0.000 | 0.164 | 0.171 |
| **OMOPSO** | -0.084 | -0.057 | -0.470 | -0.164 | 0.000 | 0.016 |
| **Random** | -0.096 | -0.070 | -0.457 | -0.171 | -0.016 | 0.000 |

The effect sizes show no mentionable difference between the *AbYSS* and *GDE3* population (compare: Table 10 and Table 11) in both analyses. However, the effect size increased from -0.004 in the analysis at iteration 50,000 to -0.029 in the analysis that included multiple snapshots between iteration 0 - 50,000. Similar effects could be observed in other pair comparisons e.g. NSGAII vs. AbYSS.

In conclusion, this section demonstrated that the inclusion of multiple performance snapshots is a feasible technique to consider the development of the search instead of just relying on an arbitrary time point in the search. However, the example showed that the inclusion of multiple snapshots only leads to marginal impacts on the descriptive statistics and effect size measures. These results have been confirmed with other examples that showed similar effects.

## 5.8 *Summary*

This chapter described the functionality and employment of the implemented multi-objective evaluation framework. The evaluation framework creates performance snapshots at multiple time points of the search, based on the user-defined analysis configuration. A performance snapshot consists of the calculation of multiple *optimal Pareto-Front* performance indicators. The novel evaluation framework enables the consolidation and slicing of performance snapshots that have been collected across different *optimal Pareto-Front* approximations. The slicing is conducted based on the specification of *MOEA* tuning parameters and a range of performance snapshot time points. Hence, the slicing enables the evaluation of the impact of different search parameter configurations on the search performance, the consolidation of a tuning parameter that features different attributes to evaluate the general performance of another tuning parameter, and the consolidation of multiple performance time points to include the development of the search in the performance evaluation.

The evaluation framework also conducts statistical analysis of the solution sets based on the consolidation of the calculated performance snapshots into slicing populations. Descriptive statistics are calculated for the sliced populations. Parametric or non-parametric significance tests are conducted based on the fulfilment of normality distribution characteristics to reveal statistical differences between pairs of sliced performance snapshot populations. Additionally, the effect sizes of the individual sliced population pairs are calculated to enable justified statements to be made regarding the difference of performance of the sliced populations.

# 6 Application of the Multi-Objective Evaluation Framework

This section demonstrates the application of the developed multi-objective evaluation framework in the problem domain of architecture reconstruction. In these experiments the performance of a set of six *MOEA* (*AbYSS*, *GDE3*, *MOEAD*, *NSGAII*, *OMOPSO*, *Random*) implementations across five different software systems (*Apache Ant v.1.9.2, Apache Commons Math v.3.2, Apache Log4j v.1.2.17, Lucene v.4.4.0, Rearchitecturer*) is evaluated. The following set of eight selected optimisation goals is applied in these experiments:

1) maximise cohesion within subsystems
2) minimise efferent coupling within subsystems
3) minimise afferent coupling within subsystems
4) converge NCCD within subsystems to 1.0
5) minimise distance in subsystems
6) minimise number of architecture violations on compilation unit level
7) minimise number of cycles on package level
8) minimise the range of compilation units in subsystems.

These selected objectives have been identified and discussed in the literature review (compare: section 2.3). An exception is the *range of compilation units in subsystems* metric, which is not a software engineering metric as such. However, it has been found through the course of this research that there was a tendency of the optimisation approaches to organise low-level artefacts into a small number of high-level artefacts to reach good performance in the number of cycles on package level, number of architecture violations on compilation unit level and coupling metrics. The other metrics have not been able to counteract this movement. As a result, the employment of the original seven software engineering optimisation goals led to an unacceptable number of solutions that featured an organisation of most low-level artefacts in only a few big

high-level artefacts. The range of compilation units in subsystems metric has been introduced to counteract this tendency. However, it is acknowledged that the minimisation of the range of compilation units in subsystems metric is not an architecture design metric that would usually be seriously considered in the *manual* design process of software architecture configurations.

The following sections discuss the findings derived from the conducted experiments. The experiments to collect the solution sets and run the analysis have been executed on two virtual machines. Appendix A: gives details of the configuration of these virtual machines.

## 6.1 *MOEA Parameter Tuning*

The *MOEA* implementations that are utilised in the *Rearchitecturer* component enable the tuning of a variety of optimisation parameters (e.g. population size, number of iterations, different crossover and variation operators and the corresponding crossover and mutation parameter settings).

The *MOEA* parameters that are applied in the evaluation experiment have been tuned iteratively until a configuration has been identified that demonstrates results sufficient to support valid conclusions on the feasibility of the developed approach. Experiments have shown that good convergence of the applied *MOEA* implementations is usually achieved within 3,000 – 5,000 iterations (compare: Figure 42 on page 175). That said, 50,000 iterations is used as a termination criterion in the following experiments and hence is more than sufficient to show that good convergence has indeed been reached. Additionally, it has been found that the population size does not impact the search outcome significantly. Nevertheless, for the sake of completeness a population size of 50 is consistently applied in the following experiment settings. Additionally, a set of crossover and mutation variation operators had to be selected and tuned. The encoding in this research employs real value decision variables (compare: section: 4.1.3). Hence, variation operators have to be applied that allow handling real

value decision variables. The applied variation operators are *Polynomial Mutation (PM)* and *Simulated Binary Crossover (SBX)*.

*PM* simulates the offspring distribution of binary-encoded bit-flip mutation on real-valued decision variables (Deb, 2001). *Polynomial Mutation (PM)* features rate and distribution index tuning parameters. The *PM* rate defines the probability that a decision variable is perturbed. The *PM* distribution index determines the magnitude of a mutation (Deb, 2001). Lower *PM* distribution index values are considered to create stronger mutations and results in decision variables that feature a more developed deviation from the original value of the decision variable. Higher values result in mutations that generate decision variable values that are similar to the parent.

*SBX* attempts to simulate the offspring distribution of binary-encoded single-point crossover on real-valued decision variables (Deb & Agrawal, 1995). The tuning parameters of *SBX* are a distribution index and a probability rate. The *SBX* rate defines the probability that a crossover variation is conducted. The *SBX* distribution index controls the shape of the offspring distribution. Higher values for the distribution index generate offspring closer to the parents. In general, lower *SBX* and/or *PM* distribution indexes are likely to create more spread and are fitter to escape local optima (Hamdan, 2011). However, this should depend on the structure of the problem and implementation of the problem encoding (e.g. causality and continuity of decision variable representations).

A set of experiments has been conducted to evaluate the impact of the four tuning parameters. Each variation operator parameter is iteratively changed while the remaining parameters are set to a fixed value. The following variation operator tunings have been evaluated in these experiments:

- pm.rate: 0.0, 0.1, 0.3, 0.5, 0.7
- pm.distributionIndex: 5, 10, 15, 20
- sbx.rate: 0.0, 0.2, 0.4, 0.6, 0.8, 1.0

- pm.distributionIndex: 5, 10, 15, 20

The five *MOEA* implementations (non-*Random*) that consider variation operators are employed in this setting with the five software systems discussed and the three conceptual target architecture models (compare: section 6.2). The outcome of the experiments revealed that a higher sbx.rate parameter configuration leads to improved optimal *Pareto-Front* performance and that the best performance is achieved with a sbx.rate setting of 1.0. The tuning of the remaining three variation operator parameters showed only marginal differences in effect size. The presentation of these results is omitted due to their insignificant impact on the search performance. However, the results of the experiments can be downloaded from the project webpage[41]. The following variation operator settings are applied in the remaining experiments to enable the traceability of the presented experiment results: mutation rate= 0.5, mutation distribution index = 10.0, crossover rate = 1.0 and crossover distribution index = 10.0.

## 6.2 *MOEA Performance in Multiple Architecture Reconstruction Scenarios*

As discussed earlier in the thesis it is anticipated in this research that the conceptual architecture is a target design of the system. Hence, in this experiment a pre-defined conceptual architecture model is considered and so the conceptual architecture is not discovered during the reconstruction process. Correspondingly, the reconstruction configuration employs the assignment of compilation units into the existing packages of the system and the assignment of the packages into the layers of the conceptual target architecture.

Experiments with transient and strict conceptual architecture models with a different number of layers (2, 3 and 4 layers) have been conducted in the course of this

[41] http://code.google.com/p/rearchitecturer/wiki/evaluationResults

research. The general structure of these conceptual models is based on the C2-architecture-style to support separation of concerns and high-level modularisation of the reconstructed system (Taylor et al., 2009).

As expected it has been found that the application of conceptual architecture models with a higher number of layers is a more complex search problem. Consequently, the absolute achievement of *MOEAs* in conceptual architecture target models with a lower number of layers is better. Nevertheless, it has been found that the application of a different number of conceptual architecture layers does not impact the individual *MOEA* implementations in comparison to one another. Hence, in the experiment presented in this section only conceptual architecture models that feature four conceptual layers are employed. However, three different conceptual architecture paradigms are used to evaluate the performance of *MOEA*s in different architecture reconstruction scenarios.

The first conceptual target architecture model features four transient layers in which each top layer can access any bottom layer. Figure 38 depicts the employed transient architecture model. Such an architecture design is common when no distribution of the system is evident and consequently all the artefacts are available on the same machine.

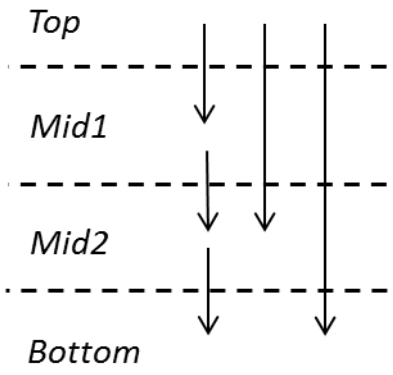


**Figure 38: Transient Architecture Model**

The second conceptual target architecture model features four strict layers in which each top layer can access only one directly depending layer. Figure 39 depicts the employed strict architecture model. Such an architecture design can, for example, be used to model a distribution of layers across machine boundaries.

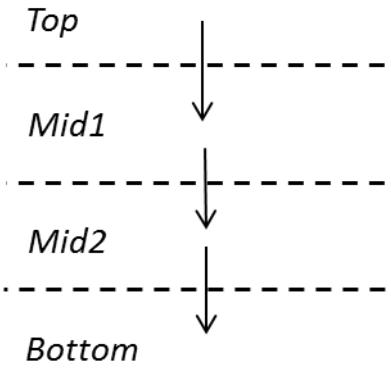


**Figure 39: Strict Architecture Model**

In the third reconstruction scenario, the strict target conceptual architecture as presented in Figure 39 is utilised and at the beginning of each seed three packages are assigned randomly to each layer. The packages and compilation units that are included in the corresponding packages are not reassigned. Hence, every solution features the initial package assignment of the corresponding seed. The idea of this reconstruction scenario is to simulate that the development stakeholder might have an understanding of the assignment of *some* packages and would like to include this knowledge in the created solutions.

These experiment scenarios add up to a total of 90 different experiment configurations (6 *MOEA* implementations x 5 systems x 3 reconstruction scenarios) that are executed for the collection of the data. Each search configuration is executed 10 times to accommodate the probability characteristics of the *MOEA* implementations. Hence, the described experiment configuration features a total of 900 solution sets.

The classification of the evaluation systems into the different conceptual target architectures features different levels of complexity. For example, the resolution of architecture violations is harder within a strict architecture than for classification into the transient target architecture. Additionally, the solution space is constrained in the reconstruction scenario, in which packages are fixed into layers. Hence, finding solutions that feature good performance in the subsystem structure metrics is more complex. To demonstrate the impact on the different target architectures the dataset is sliced by consolidating algorithms and systems and the separation of result sets is based on the different target architecture designs. The analysis of the sliced datasets confirmed a different level of performance achievement depending on the level of the complexity of the reconstruction scenario. Figure 40 depicts the corresponding *Hypervolume* performance development graph:

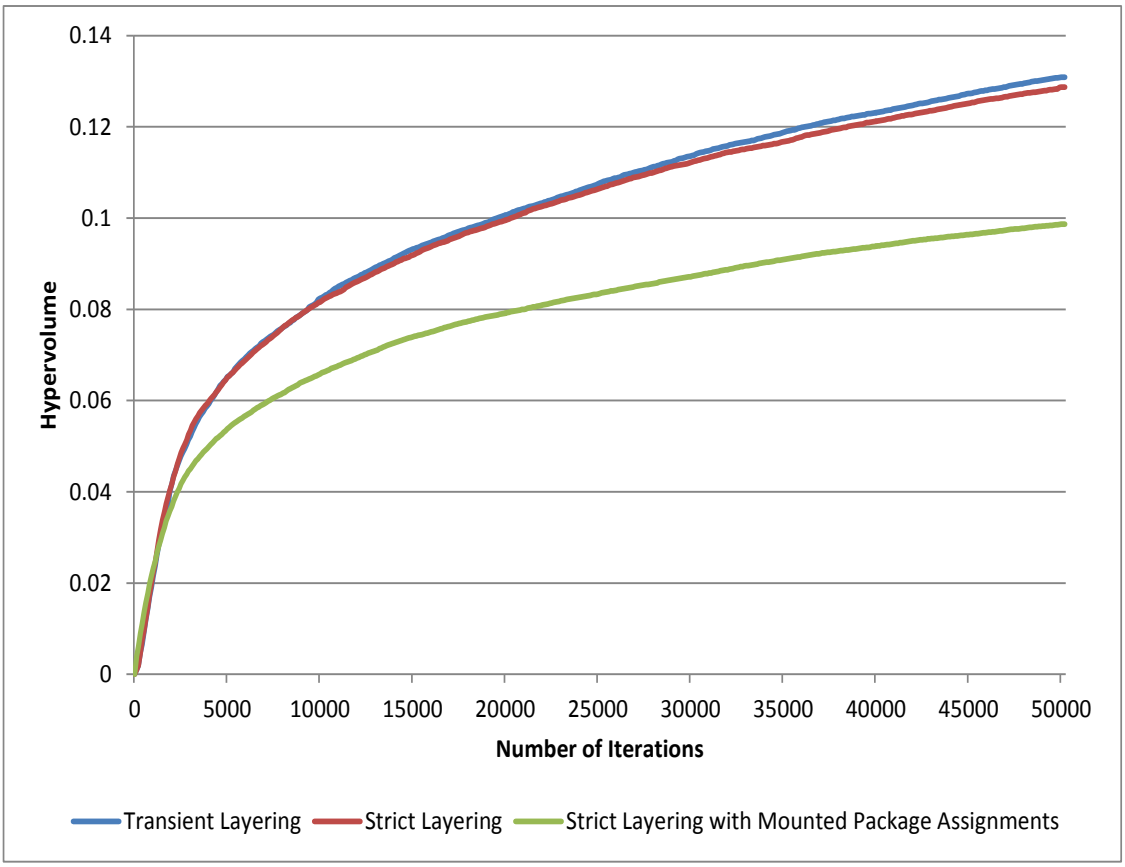


**Figure 40: Hypervolume - Slicing based on Conceptual Target Architecture**

The graph shows that the transient- and strict- reconstruction scenarios achieve equally good *Hypervolume* performance. Understandably, the *Hypervolume* performance of the third architecture example is substantially lower due to the increase of complexity based on the upstream assignment and mounting of packages into subsystems.

In the analysis presented in section 6.3 and 6.4 the performance of the employed *MOEA* implementations across the three reconstruction scenarios is evaluated. The three architecture reconstruction datasets are consolidated to enable such an analysis. However, a prerequisite for the validity of such a comparison is that each reconstruction scenario contributes to the approximated true *Pareto-Front,* as the analysis relies mainly on achievement in the objective space and convergence of the optimal *Pareto-Fronts*. The solution sets of a reconstruction scenario are excluded from the analysis if the optimal *Pareto-Front* of that reconstruction scenario is not contributing to the approximated true *Pareto-Front* and the slicing is based on the employed *MOEA* implementations. For example, if the least complex architecture reconstruction scenario dominates the *optimal Pareto-Front* of the more complex architecture scenarios the analysis only considers the solution sets of the least complex architecture reconstruction solution sets. The *Contribution* performance metric is useful in this regard to confirm if each reconstruction scenario is able to contribute towards the approximated true *Pareto-Front.* Figure 41 depicts the development of the contribution performance metric through the search process in the individual reconstruction scenarios.

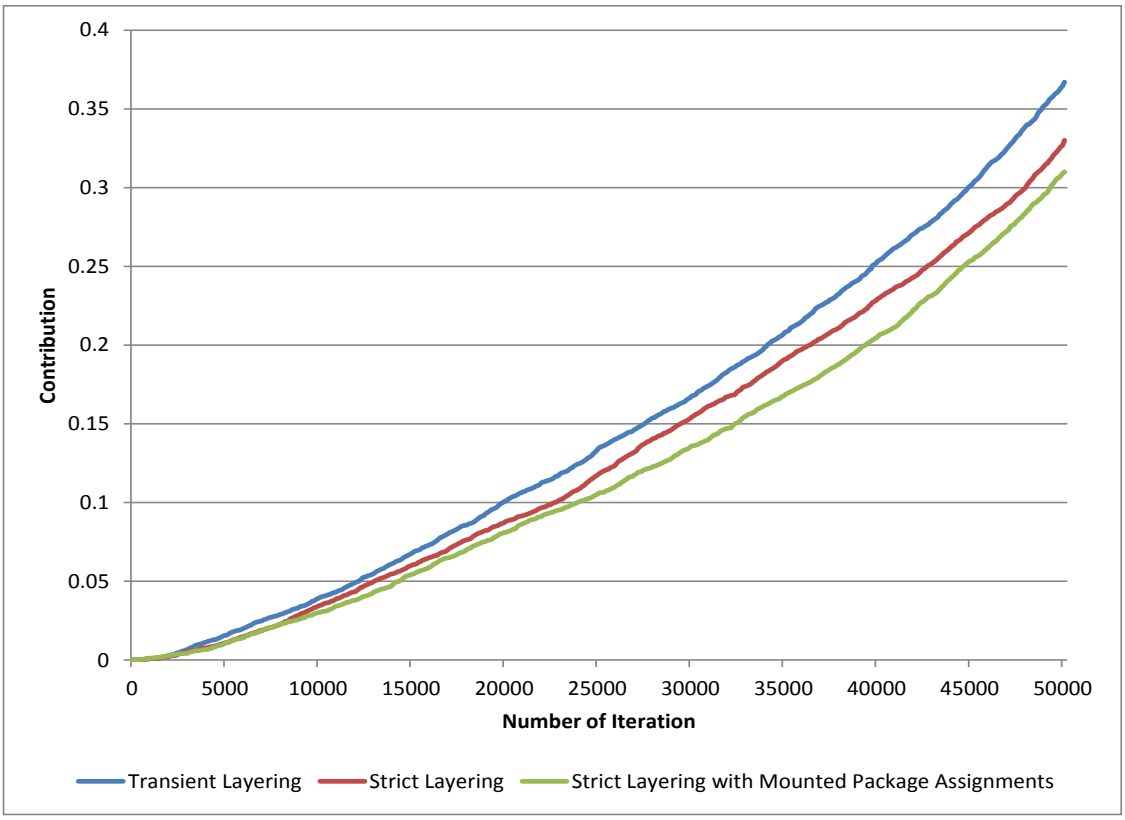


**Figure 41: Contribution - Slicing by Conceptual Target Architecture**

The contribution graph depicts that the complexity of the search impacts the *Contribution* outcome. However, the optimal *Pareto-Fronts* of the individual reconstruction scenarios feature a fairly equal contribution to the approximated true *Pareto-Front* despite the different complexity evident in the individual reconstruction scenarios. Hence, it can be concluded that the analyses of the *MOEA* implementations is representative for all three employed reconstruction scenarios.

## 6.3 *Analysis of MOEA Performance in the Objective Space*

The main objective of this research is the application of multi-objective optimisation techniques in the application domain of architecture reconstruction. High-level architecture design metrics are employed as objectives to implement this research project. Such high-level architecture design metrics have not been employed in related research efforts. The review of the capability of the employed optimisation techniques

to advance the individual objective dimensions is necessary to enable statements on the applicability of the selected high-level architecture design metrics to be made.

No generally accepted method has been established in other multi-objective research to assess the achievement in the objective space. For example, related research has relied on the presentation of the *MQ* measure, *Generational Distance* and *Error Ratio* to evaluate the relative convergence of optimal *Pareto-Fronts*. However, within the present approach the explicit reliance on the application of multi-objective performance metrics that calculate relative convergence of similarity to a best optimal *Pareto-Front* might be misleading. For example, two optimisation configurations might reduce the number of architecture violations to 500 and 800. A convergence-based performance metric will confirm a better performance for the approach that achieved 500 architecture violations if we ignore the existence of other solutions and objectives in this example. However, both solutions are most likely still too complex to allow stakeholders to understand the solution and manually resolve remaining forbidden dependencies. Hence, *both* configurations would be infeasible for use in the target application domain. Hence, assessment of the objective achievement is an important component to assess the overall feasibility of the developed approach.

Theoretically, the objective achievement can be assessed based on any of the solution sets evident in a multi-objective optimisation. These kinds of solutions sets are: the complete set of visited solutions, the current population, and the optimal *Pareto-Front*. The implemented evaluation framework supports the assessment of the objective achievement based on all three of these solution sets. Descriptive statistics are calculated for the individual solution sets at each performance snapshot. The selection of the solution set and the descriptive statistics depends on the user's research interest.

The first analysis demonstrates the convergence in the individual objectives. Hence, the solutions of the generational populations or more specifically the solution that features the best performance in the desired objective is used in this analysis.

This evaluation uses the same data configuration as described in section 6.2 with only one variation operator setting. The results are sliced by the applied *MOEA* implementation. Correspondingly, a population of 150 is created for each *MOEA* implementation at each performance snapshot. The evaluation framework reports descriptive statistics for each performance snapshot. In this evaluation, the interval for performance snapshots is aligned with the population size of the generations and is correspondingly set to 50 iterations. Hence, 1,000 performance snapshots are calculated in this analysis.

Figure 42 depicts the mean development of the best achievement of the eight employed objectives. The mean for each objective is presented for each performance snapshot.

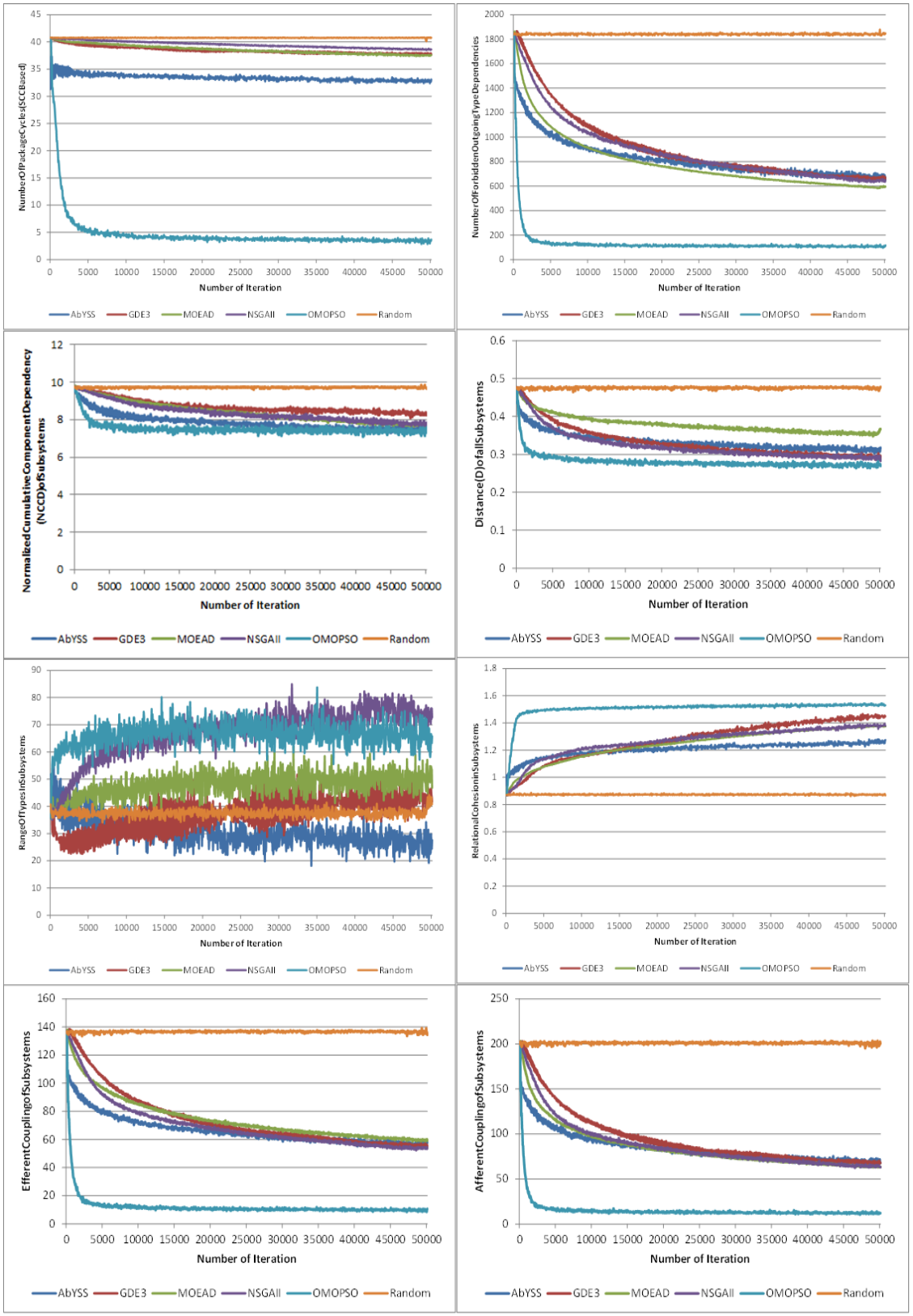


**Figure 42: Objective Achievement in Populations**

The objective development depicts that each of the employed *MOEA* implementations features better performance than *Random* in seven of the eight objective dimensions. Additionally, *OMOPSO* features the best overall advancement in the same seven objectives. Furthermore, no specific performance differences can be reported between *NSGA-II*, *GDE3*, *ABYSS* and *MOEAD* in these seven objective dimensions.

An exception is the *Range Of Types In Subsystems* objective in which only *AbYSS* shows better convergence than *Random*. However, it needs to be considered that the presented results are the mean of the best performances per population at each generation of the search. Hence, it is not a representation of the overall search process or achievement in that objective. The presented results are simply an indication of the feasibility of the approach to converge the individual objectives.

It may be suggested that the review of the objective achievement of the optimal *Pareto-Front* is a better evaluation instrument to assess the general feasibility of an optimisation approach in the individual objective dimensions. However, the presentation of the mean development of the objective progress of the optimal *Pareto-Front* solution set is most likely also an unreliable means to assess the performance of a configuration setting. The reason for this is that the trade-off concept of optimal *Pareto-Fronts* leads to the inclusion of solutions that dominate any area of the objective space. These solutions therefore might feature poor performance in the reviewed objective. Hence, while a good progress in the minimum and maximum value of an objective is achieved the mean progress might be relatively constant in the reviewed objective. Hence, it is suggested in this research that the reporting of descriptive statistics, and in particular the review of minimum, maximum and distribution characteristics of objective achievement of the optimal *Pareto-Front,* are more valuable methods to review achievement in the objective space.

This analysis relies on achievement in the objective space in the optimal *Pareto-Fronts*. The normality condition is not fulfilled in these datasets. Hence, no valid conclusions can be drawn on the distribution of the data based on the *Mean* and *SD* measures. Hence, *Mean* and *SD* measures are not presented but can be found in the original datasets on the project webpage. Consequently, the following tables present the *Minimum*, *Maximum* and *Median* of the objective measures of the optimal *Pareto-Fronts.* The populations for these descriptive measures are created based on the objective measures of the solutions of the optimal *Pareto-Fronts* of the created slices. The results presented in the tables are sliced by the applied MOEA implementation and system to enable the assessment of the objective performance of *MOEAs* in the individual systems.

Histograms are presented to communicate the distribution characteristics and the achievement in the objective space. In the first presented objective (*Number of Package Cycles*) the histograms are presented for each system individually (see: Figure 43). The review of the performance of the individual *MOEA* implementations in the individual systems has shown that the *MOEA* implementations feature similar performance characteristics in the other objectives. Hence, in the histograms of the remaining seven objectives the optimal *Pareto Fronts* of all systems are consolidated in a single histogram for each objective. It is suggested that such a consolidation is admissible as the aim of this section is to show the general performance of *MOEA* implementation in the objective space across a set of representative software systems. It is acknowledged that the different complexity of the systems might lead to the aggregation of objective values at different objective regions. Nevertheless, the datasets are still comparable as the same evaluation systems are used in the slices.

Furthermore, the three architecture reconstruction scenarios are consolidated to convey a more representative demonstration of the performance of the *MOEA* implementations in the objective space across different systems. No noteworthy difference between the reconstruction scenarios has been found that would justify the

separate presentation by reconstruction scenario (compare: section 6.2). An aim of this analysis is to show the distribution characteristics of the optimal *Pareto-Front* sets. This is useful to assess the number of solutions that are discovered within promising areas of the search. Hence, the optimal *Pareto-Fronts* of the seeds of each slice are not merged to include the distribution characteristics of the different configuration settings. The total number of individual optimal *Pareto-Fronts* that are considered in this analysis is 900 (compare: section 6.2). These optimal *Pareto-Fronts* feature a combined total of 236,570 solutions. The data sets that feature a slicing based on system, employed reconstruction scenario and also the number of non-dominated solutions to each slice are available on the project webpage.

Table 12 presents the minimum, maximum and median of the *Number of Packages Cycles* metric in the optimal *Pareto-Fronts*.

**Table 12: Descriptive Statistics – *Number of Package Cycles***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 46.00 | 60.00 | 55.95 |
| | GDE3 | 55.00 | 60.00 | 59.70 |
| | MOEAD | 39.00 | 60.00 | 58.17 |
| | NSGAII | 55.00 | 60.00 | 59.97 |
| | OMOPSO | 0.00 | 60.00 | 35.58 |
| | Random | 55.00 | 60.00 | 59.99 |
| **apache log4j** | AbYSS | 1.00 | 620.00 | 370.05 |
| | GDE3 | 1.00 | 620.00 | 358.46 |
| | MOEAD | 1.00 | 590.00 | 287.46 |
| | NSGAII | 1.00 | 620.00 | 332.29 |
| | OMOPSO | 0.00 | 600.00 | 347.77 |
| | Random | 180.00 | 610.00 | 477.19 |
| **apache math** | AbYSS | 44.00 | 60.00 | 52.54 |
| | GDE3 | 50.00 | 60.00 | 57.91 |
| | MOEAD | 47.00 | 60.00 | 57.51 |
| | NSGAII | 55.00 | 60.00 | 58.24 |
| | OMOPSO | 0.00 | 60.00 | 25.62 |
| | Random | 55.00 | 60.00 | 55.02 |
| **lucene** | AbYSS | 15.00 | 20.00 | 19.20 |
| | GDE3 | 19.00 | 20.00 | 19.59 |
| | MOEAD | 18.00 | 20.00 | 19.60 |
| | NSGAII | 19.00 | 20.00 | 19.52 |
| | OMOPSO | 0.00 | 20.00 | 14.69 |
| | Random | 19.00 | 20.00 | 19.46 |
| **rearchitecturer** | AbYSS | 29.00 | 47.00 | 41.05 |
| | GDE3 | 34.00 | 47.00 | 42.99 |
| | MOEAD | 32.00 | 47.00 | 43.82 |
| | NSGAII | 37.00 | 47.00 | 44.61 |
| | OMOPSO | 0.00 | 47.00 | 26.12 |
| | Random | 39.00 | 47.00 | 44.75 |

The optimal *Pareto-Front* development for the *Number of Package Cycles* objective depicts that *OMOPSO* features the best performance based on the achieved range in all systems. Additionally, only *OMOPSO* is able to discover solutions in all systems that feature zero cycles. The other *MOEA* implementations can only compete with *OMOPSO* in terms of absolute achievement in the *log4j* system. The lowest optimal Pareto-Front

median value is achieved by the OMOPSO implementation in four of the five systems. The only exception is the decomposition based *MOEAD* implementation in the *log4j* system. Hence, the absolute achievement towards the desired optimisation direction of the other *MOEA* implementations depends strongly on the complexity of the system.

Figure 43 presents the histograms of the *Number of Package Cycles* objective in the migrated evaluation systems. The number of solutions of the optimal *Pareto-Fronts* for each algorithm between an objective value range is shown in the histograms.

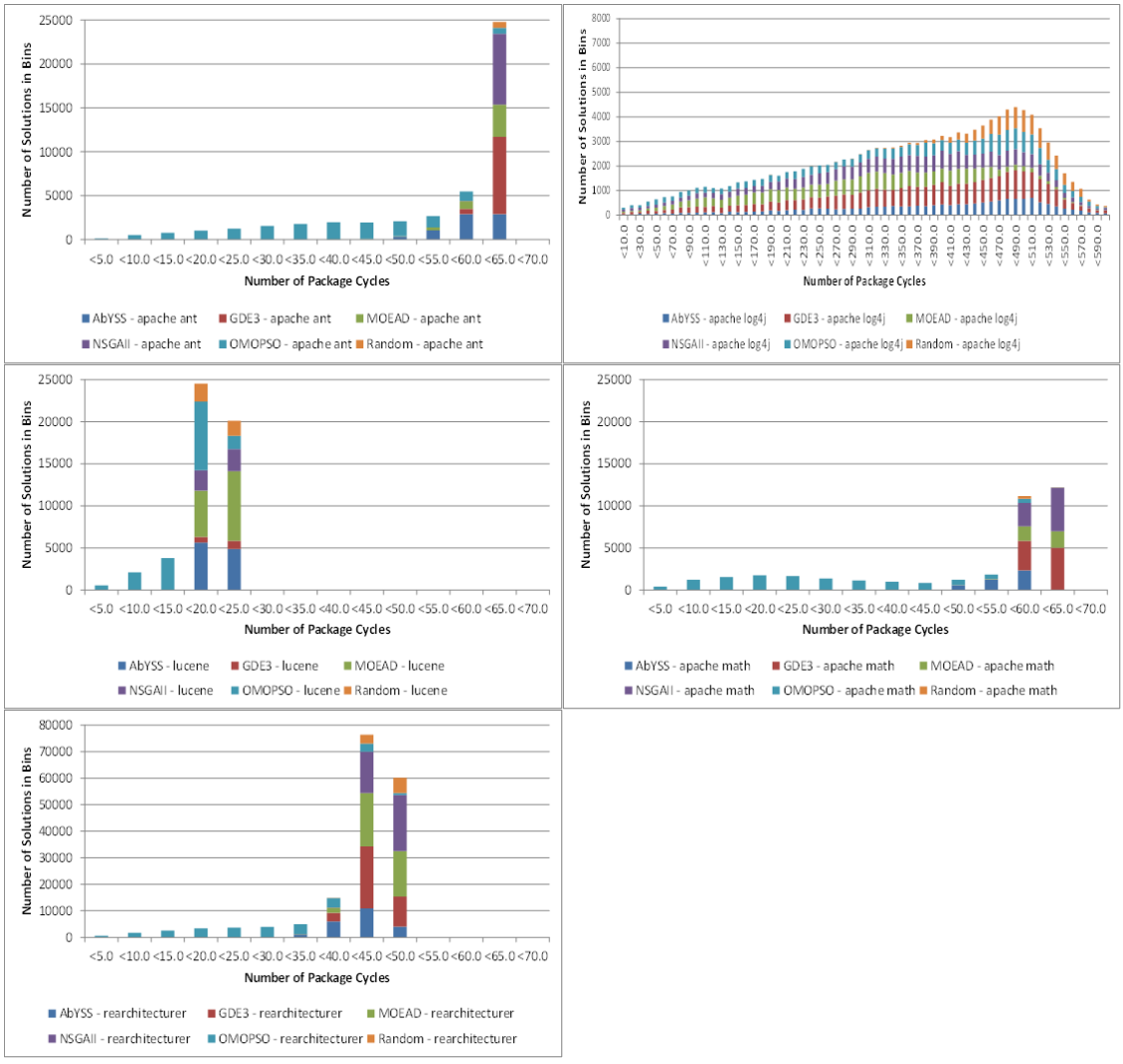


**Figure 43: Histogram Number of Package Cycles Objective in Systems**

The histograms depict that the *apache log4j* system features the most homogenous distribution and the widest spread, with solutions that feature up to 620 cycles (compare: Table 12). A possible explanation is that *log4j* features the lowest complexity in terms of number of types and dependencies. Hence, the employed *MOEA* implementations can visit a relatively bigger part of the search space in the *log4j* system. Consequently, solutions can be discovered that feature better achievement in the objectives. This most likely also leads to higher trade-offs in other objective and consequently a wider range of solutions. The other systems also feature a high number of solutions with acceptable *Number of Cycles in Packages* measures. It is not really possible to state at this stage what are the reasons why there is such a narrow range achieved in the more complex systems and why only *OMOPSO* is able to find highly promising solutions in the more complex systems.Table 13 depicts the descriptive statistics of the *Number of Forbidden Outgoing Type Dependencies* metric in the optimal *Pareto-Fronts*.

**Table 13: Descriptive Statistics - *Number of Forbidden Type Dependencies***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 460.00 | 3000.00 | 2033.37 |
| | GDE3 | 320.00 | 3000.00 | 1784.63 |
| | MOEAD | 220.00 | 3000.00 | 1524.63 |
| | NSGAII | 500.00 | 3000.00 | 1770.56 |
| | OMOPSO | 1.00 | 3000.00 | 1656.58 |
| | Random | 2000.00 | 3000.00 | 2711.85 |
| **apache log4j** | AbYSS | 11.00 | 19.00 | 16.46 |
| | GDE3 | 13.00 | 19.00 | 17.49 |
| | MOEAD | 12.00 | 19.00 | 17.44 |
| | NSGAII | 13.00 | 19.00 | 17.54 |
| | OMOPSO | 0.00 | 19.00 | 12.57 |
| | Random | 15.00 | 19.00 | 17.64 |
| **apache math** | AbYSS | 660.00 | 3000.00 | 2119.18 |
| | GDE3 | 330.00 | 3000.00 | 1953.53 |
| | MOEAD | 600.00 | 3000.00 | 1909.11 |
| | NSGAII | 400.00 | 3000.00 | 1910.24 |
| | OMOPSO | 1.00 | 3000.00 | 1458.10 |
| | Random | 0.00 | 3000.00 | 2792.83 |
| **lucene** | AbYSS | 2.00 | 2500.00 | 1526.55 |
| | GDE3 | 1200.00 | 2500.00 | 2076.50 |
| | MOEAD | 70.00 | 2500.00 | 1321.62 |
| | NSGAII | 850.00 | 2500.00 | 1895.28 |
| | OMOPSO | 1.00 | 2500.00 | 1474.07 |
| | Random | 980.00 | 2500.00 | 1936.06 |
| **rearchitecturer** | AbYSS | 130.00 | 930.00 | 636.45 |
| | GDE3 | 10.00 | 930.00 | 565.15 |
| | MOEAD | 140.00 | 890.00 | 523.92 |
| | NSGAII | 6.00 | 920.00 | 541.79 |
| | OMOPSO | 1.00 | 900.00 | 532.02 |
| | Random | 580.00 | 910.00 | 783.12 |

In general, the presented statistics of the optimal *Pareto-Fronts* of the *Number of Forbidden Type Dependencies* achievement show results similar to those achieved for the previous metric. In general, *OMOPSO* features the best absolute achievement in this objective across all systems. An exception is *Random* that discovered a solution with 0.0 forbidden type dependencies in the *apache math* system. Additionally, *NSGAII* and

*GDE3* are able to find promising solutions in the *Rearchitecturer* system and *AbYSS* is able to find promising solutions for the *lucene* system.

The other *MOEA* algorithms show worse performance in more complex systems apart from this coincidental discovery by the *Random* implementation in the apache math system. A difference in comparison to the findings of the previous *Number of Package Cycles* metric is the small range of metric measurements in the *log4j* system.

Figure 44 presents the histogram of the *Number of Forbidden Outgoing Type Dependencies* objective. From this objective onwards the individual system slices are consolidated in one histogram. The reason for this is that this study mainly investigates the general performance of the individual *MOEA* implementations in a representative set of software systems. Nevertheless, the tables still give the descriptive statistics for the individual systems to enable the assessment of the impact of different system sizes on the objective achievement.

It is anticipated in the *Number of Forbidden Outgoing Type Dependencies* objective that solutions with more than 100.0 solutions can be considered as infeasible. Hence, solutions with a higher number of *Forbidden Outgoing Type Dependencies* are consolidated in the last bin.

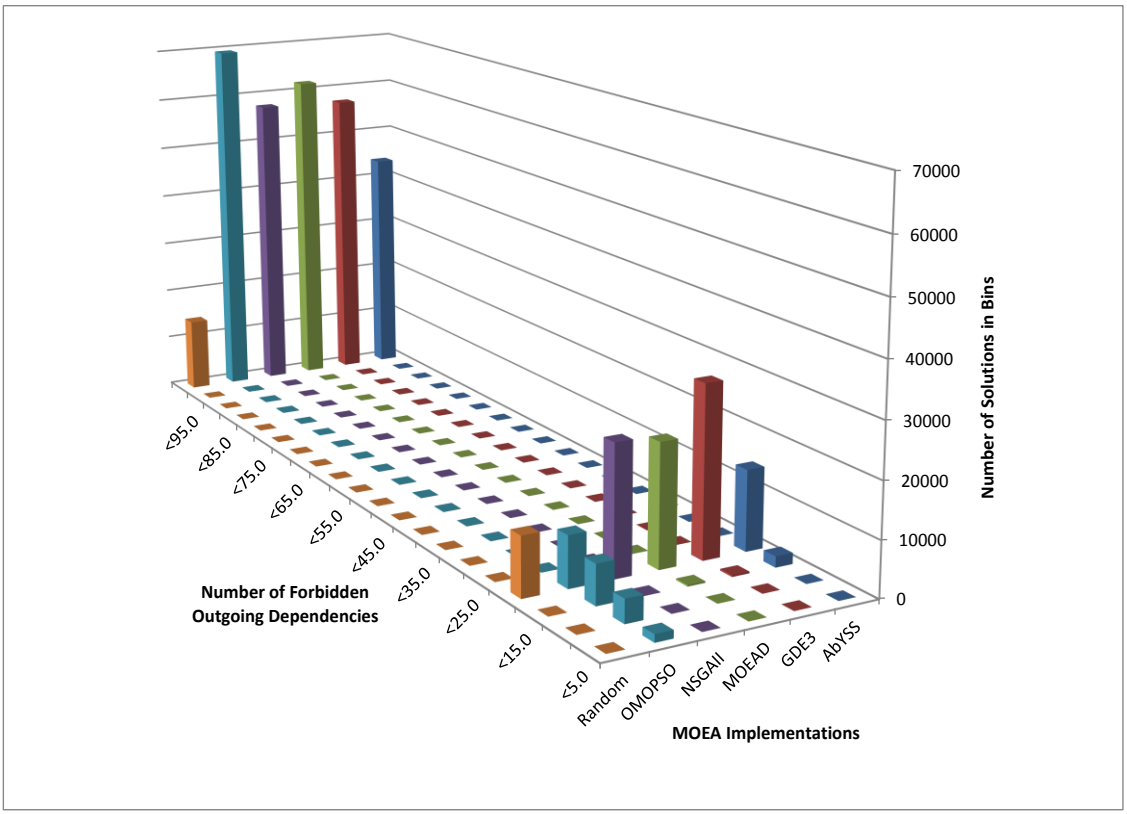


**Figure 44: Histogram *Number of Forbidden Outgoing Dependencies objective***

Figure 44 depicts that *OMOPSO* is able to progress the search to solutions with a reasonable number of solutions with fewer than 15.0 architecture violations. Such solutions are able to be discovered in all systems (compare: Table 13). The solutions of the non-*OMOPSO* slices are probably discovered in the *apache-log4j* systems as the non-*OMOPSO* algorithms have been unable to discover solutions with fewer than 100.0 architecture violations in the more complex software systems (compare: Table 13). This assertion is also supported by the flat landscape between 25.0 and 100.0 violations, in which barely any solutions are discovered by any of the employed *MOEA* implementations. Consequently, the data supports the expectation that the complexity of the transformed system strongly impacts achievement in the *Number of Forbidden Outgoing Type Dependencies.*

The third objective (relating to NCCD) measures the internal structural quality of the subsystems of the desired target architecture. Table 14 illustrates the *Minimum*, *Maximum* and *Median* of the NCCD metric.

**Table 14: Descriptive Statistics - NCCD**

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 14.00 | 170.00 | 29.62 |
| | GDE3 | 15.00 | 170.00 | 28.34 |
| | MOEAD | 14.00 | 85.00 | 24.14 |
| | NSGAII | 12.00 | 180.00 | 29.25 |
| | OMOPSO | 6.20 | 300.00 | 40.44 |
| | Random | 21.00 | 40.00 | 22.29 |
| **apache log4j** | AbYSS | 0.00 | 1.50 | 0.69 |
| | GDE3 | 0.00 | 1.50 | 0.69 |
| | MOEAD | 0.00 | 1.70 | 0.83 |
| | NSGAII | 0.00 | 1.60 | 0.72 |
| | OMOPSO | 0.00 | 1.30 | 0.71 |
| | Random | 0.00 | 1.20 | 0.66 |
| **apache math** | AbYSS | 0.00 | 85.00 | 7.49 |
| | GDE3 | 3.50 | 80.00 | 6.84 |
| | MOEAD | 4.10 | 12.00 | 6.03 |
| | NSGAII | 2.80 | 85.00 | 6.92 |
| | OMOPSO | 0.00 | 100.00 | 8.25 |
| | Random | 0.00 | 7.30 | 5.98 |
| **lucene** | AbYSS | 3.30 | 180.00 | 10.99 |
| | GDE3 | 7.50 | 18.00 | 10.36 |
| | MOEAD | 5.40 | 26.00 | 9.85 |
| | NSGAII | 7.20 | 22.00 | 10.31 |
| | OMOPSO | 2.70 | 330.00 | 12.60 |
| | Random | 7.00 | 19.00 | 10.29 |
| **rearchitecturer** | AbYSS | 0.00 | 70.00 | 6.40 |
| | GDE3 | 0.00 | 85.00 | 6.67 |
| | MOEAD | 3.20 | 35.00 | 5.50 |
| | NSGAII | 1.90 | 90.00 | 6.70 |
| | OMOPSO | 0.00 | 95.00 | 8.04 |
| | Random | 0.00 | 10.00 | 5.92 |

The minimum values indicate that the approach is capable of optimising the *NCCD* objective towards promising values. In general, the results are similar to the results for the previous objectives with the best minimum value and range achieved by *OMOPSO* across all systems. *Random* is again able to discover solutions in the *Rearchitecturer*, *Apache Math*, and *Apache Log4j* systems that feature a *NCCD* measure of 0.00. The review of the distribution in the histogram reveals that these discoveries of the *Random* implementation are of a coincidental nature, as Random only contributes a few solutions with such competitive NCCD measures to the optimal *Pareto-Fronts*. Figure 45 presents the histogram of the *NCCD* objective. Solutions with a *NCCD* measure of more than 20.0 are consolidated in the last bin.

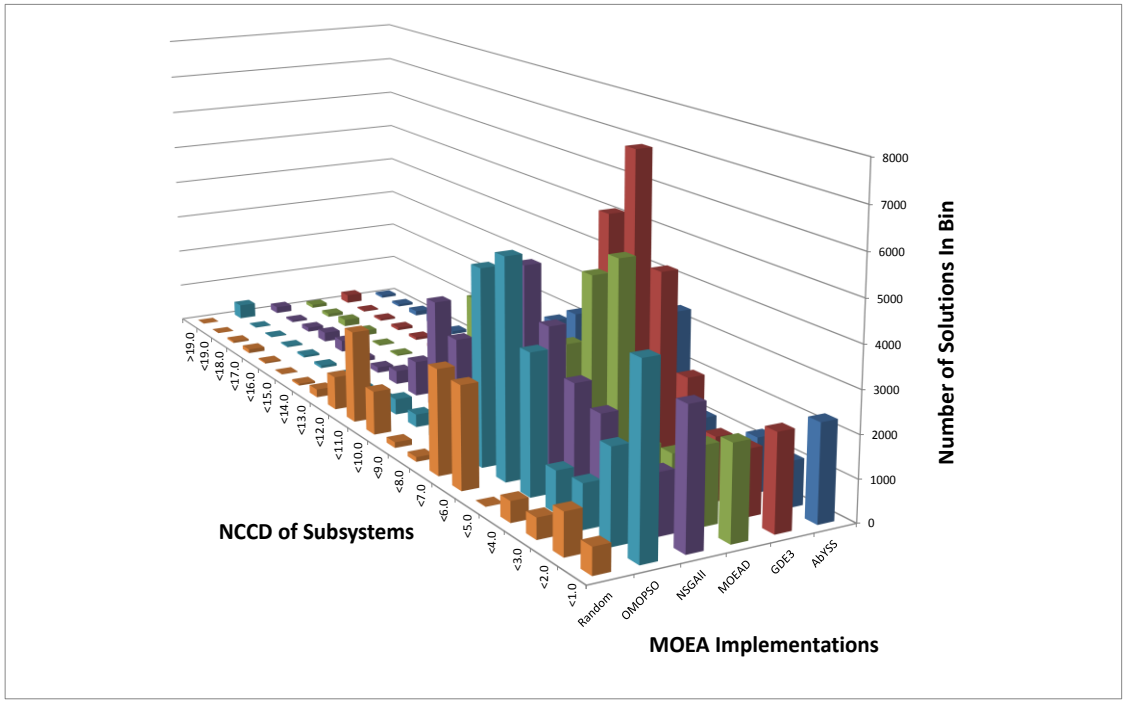


**Figure 45: Histogram NCCD objective**

The histogram indicates that all of the algorithms are able to achieve solutions with acceptable *NCCD* measures. In section 2.3.4 it has been discussed that an optimal *NCCD* value is 1.0. Hence, one could argue that some of the solutions are over-optimised by the *MOEA* implementations. Almost all of the solutions of the analysed optimal *Pareto-*

*Front* datasets feature a *NCCD* measure of < 15.0. A *NCCD* of less than 15.0 is an acceptable structural quality based on the complexity of the optimised software systems. Hence, this raises the question if the *NCCD* metric is a good quality indicator that challenges the optimisation adequately. The optimal *Pareto-Fronts* contain solutions that feature the best trade-offs between the individual objectives. Hence, many other solutions have been visited that feature worse quality in the *NCCD* metric. Hence, very good achievement in a substantial part of the optimal *Pareto-Front* is certainly good. However, techniques are required, as presented in section 4.2.2, to identify the most promising solutions based on other solution aspects.

The fourth objective (*Distance from Main Sequence*) measures the mixture of abstract and implementation classes in high-level artefacts. Table 15 depicts the Minimum, Maximum and Median of the *Distance from the Main Sequence* metric.

**Table 15: Descriptive Statistics - *Distance from the Main Sequence***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 0.26 | 0.74 | 0.48 |
| | GDE3 | 0.25 | 0.73 | 0.46 |
| | MOEAD | 0.24 | 0.62 | 0.46 |
| | NSGAII | 0.24 | 0.76 | 0.45 |
| | OMOPSO | 0.02 | 0.80 | 0.46 |
| | Random | 0.48 | 0.60 | 0.53 |
| **apache log4j** | AbYSS | 0.00 | 95.00 | 48.74 |
| | GDE3 | 0.00 | 90.00 | 45.54 |
| | MOEAD | 0.00 | 85.00 | 38.58 |
| | NSGAII | 0.00 | 90.00 | 42.88 |
| | OMOPSO | 0.00 | 90.00 | 45.74 |
| | Random | 0.00 | 90.00 | 60.69 |
| **apache math** | AbYSS | 0.00 | 0.69 | 0.50 |
| | GDE3 | 0.22 | 0.72 | 0.48 |
| | MOEAD | 0.26 | 0.66 | 0.47 |
| | NSGAII | 0.24 | 0.77 | 0.46 |
| | OMOPSO | 0.00 | 0.77 | 0.46 |
| | Random | 0.00 | 0.61 | 0.54 |
| **lucene** | AbYSS | 0.12 | 0.70 | 0.42 |
| | GDE3 | 0.34 | 0.58 | 0.45 |
| | MOEAD | 0.24 | 0.63 | 0.41 |
| | NSGAII | 0.26 | 0.60 | 0.44 |
| | OMOPSO | 0.00 | 0.77 | 0.41 |
| | Random | 0.30 | 0.60 | 0.45 |
| **rearchitecturer** | AbYSS | 0.00 | 0.76 | 0.48 |
| | GDE3 | 0.00 | 0.77 | 0.46 |
| | MOEAD | 0.28 | 0.70 | 0.46 |
| | NSGAII | 0.18 | 0.76 | 0.46 |
| | OMOPSO | 0.00 | 0.78 | 0.46 |
| | Random | 0.00 | 0.60 | 0.51 |

All of the *MOEA* implementations achieved acceptable minimum values in the considered optimal *Pareto-Fronts*. The maximum values indicate that the *MOEA* implementations also entered highly infeasible solution space for the *log4j* system. Contrary to prior expectations, the *Distance from Main Sequence* measures in the more complex systems do not feature any extreme high values. Figure 46 presents the

histogram of the *Distance* objective. A classification of solutions into ten bins has been chosen in this histogram with a cut-off value of 2.0 to limit the visualisation to competitive solutions.

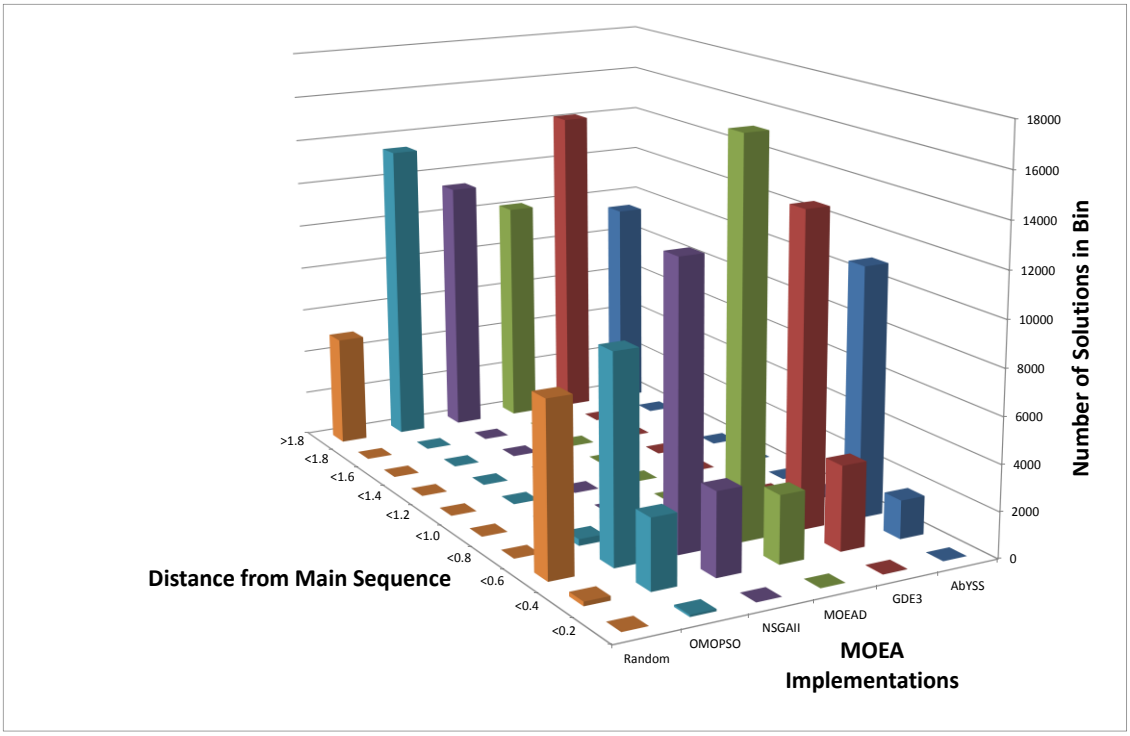


**Figure 46: Histogram *Distance from Main Sequence* objective**

The histogram shows that all of the *MOEA* implementations are able to discover promising solution candidates. As for the previous objectives *OMOPSO* still features the best absolute achievement. Nevertheless, *MOEAD, GDE3* and *AbYSS* discover a higher number of promising solutions than the *OMOPSO* search. A high number of promising solutions is good to allow stakeholders to make use of the implemented constraining approaches to identify a final solution.

It has been shown in Figure 42 that the fifth objective (*Range of Types in Subsystem* metric) features hardly any convergence towards the desired optimisation direction within the mean generational development. Table 16 illustrates the descriptive statistics of the *Range of Types in Subsystems* metric.

**Table 16: Descriptive Statistics - *Range of Types in Subsystems***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 0.00 | 1000.00 | 598.52 |
| | GDE3 | 2.00 | 1100.00 | 659.80 |
| | MOEAD | 1.00 | 1100.00 | 626.73 |
| | NSGAII | 0.00 | 1000.00 | 654.09 |
| | OMOPSO | 0.00 | 1100.00 | 693.04 |
| | Random | 1.00 | 490.00 | 199.07 |
| **apache log4j** | AbYSS | 0.00 | 310.00 | 136.08 |
| | GDE3 | 0.00 | 310.00 | 145.43 |
| | MOEAD | 0.00 | 290.00 | 117.32 |
| | NSGAII | 0.00 | 310.00 | 153.22 |
| | OMOPSO | 0.00 | 310.00 | 149.36 |
| | Random | 0.00 | 230.00 | 81.72 |
| **apache math** | AbYSS | 0.00 | 920.00 | 483.81 |
| | GDE3 | 1.00 | 980.00 | 515.51 |
| | MOEAD | 0.00 | 840.00 | 384.56 |
| | NSGAII | 0.00 | 930.00 | 519.53 |
| | OMOPSO | 0.00 | 960.00 | 597.23 |
| | Random | 0.00 | 470.00 | 154.07 |
| **lucene** | AbYSS | 0.00 | 1400.00 | 700.16 |
| | GDE3 | 0.00 | 1000.00 | 287.72 |
| | MOEAD | 0.00 | 1300.00 | 636.22 |
| | NSGAII | 2.00 | 1100.00 | 457.84 |
| | OMOPSO | 0.00 | 1400.00 | 714.48 |
| | Random | 2.00 | 1000.00 | 434.70 |
| **rearchitecturer** | AbYSS | 0.00 | 430.00 | 179.62 |
| | GDE3 | 0.00 | 480.00 | 214.54 |
| | MOEAD | 1.00 | 390.00 | 144.87 |
| | NSGAII | 1.00 | 460.00 | 222.36 |
| | OMOPSO | 0.00 | 490.00 | 238.61 |
| | Random | 0.00 | 260.00 | 71.81 |

The analyses of the objective achievement in the optimal *Pareto-Fronts* show that, for all algorithms, solutions are discovered that feature a small *Range of Types in Subsystems* measurement. All *MOEA* implementations are able to find solutions in all systems that feature hardly any difference in subsystem size. Stakeholders are probably interested in a relatively homogenous distribution of types in subsystems. However, an

absolute equal size of subsystems is certainly not required. Hence, a review of the distribution in this objective is valuable to allow statements to be made on the number of solutions that are discovered within the promising area of this dimension. Figure 47 presents the histogram of the *Range of Types in Subsystems* objective.

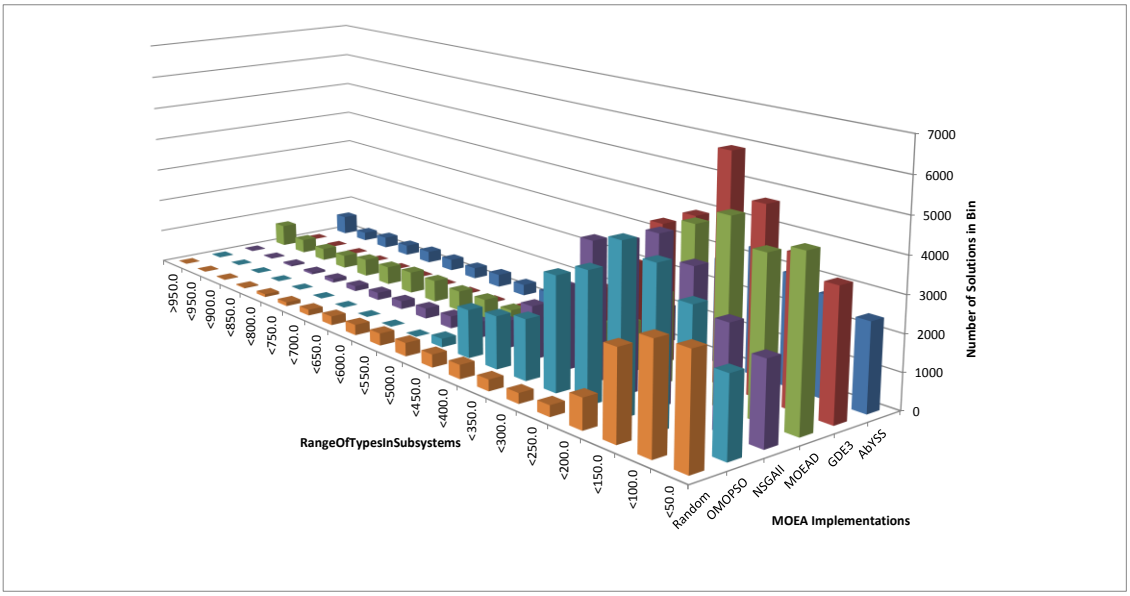


**Figure 47: Histogram Range of Types in Subsystems objective**

The discovered solutions of all *MOEA* implementations feature a wide range of *Range of Types in Subsystems* measures: solutions with balanced and unbalanced distributions of types in subsystems are discovered by all *MOEA* implementations.

Recall that the *Range of Types in Subsystems* metric is used to counteract the risk of *MOEA* implementations organising low-level artefacts into just one subsystem to achieve low numbers of cycles and architecture violations. Hence, the *Range of Types in Subsystems* metric does not express a software engineering concept as such. However, the wide spread of the objective measures shows that quite a few solutions of the optimal Pareto-Fronts are not useful for further consideration.

*OMOPSO* featured the best absolute achievement in the previously discussed objectives. This achievement was particularly relevant as only *OMOPSO* has been able to

drive the search into highly desired objective space within the more complex software systems. However, the histograms for the *Range of Types* objective depict that all *MOEA* implementations are able to discover a number of solutions with a good balanced distribution of types in the subsystem characteristics. Additionally, it can be stated that the *MOEAD* and *GDE3* implementation are even able to find a higher number of promising solutions than *OMOPSO* in this objective.

Another interesting aspect of these particular results is the competiveness of the *Random* search implementation in terms of absolute achievement as well as the number of promising solutions in this objective. A possible explanation is that the decision variables to control the classification feature are not independent of one other and no causality in the organisation of low-level artefacts in subsystems is evident. Consequently, the evolutionary concepts of the advanced *MOEA* implementations cannot be leveraged in this objective.

The sixth objective considered in this discussion is the *Relational Cohesion in Subsystems* metric. The *Relational Cohesion* metric is the only objective that features the maximisation of a metric as an optimisation target. Table 17 illustrates the descriptive statistics and Figure 48 presents the histograms of the *Relational Cohesion in Subsystems* objective.

**Table 17: Descriptive Statistics - *Relational Cohesion in Subsystems***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 0.72 | 1.30 | 0.91 |
| | GDE3 | 0.70 | 1.90 | 0.92 |
| | MOEAD | 0.80 | 1.40 | 0.94 |
| | NSGAII | 0.74 | 2.00 | 0.95 |
| | OMOPSO | 0.74 | 2.00 | 1.06 |
| | Random | 0.78 | 1.10 | 0.89 |
| **apache log4j** | AbYSS | 0.00 | 0.78 | 0.46 |
| | GDE3 | 0.00 | 0.79 | 0.45 |
| | MOEAD | 0.00 | 0.77 | 0.44 |
| | NSGAII | 0.00 | 0.78 | 0.44 |
| | OMOPSO | 0.00 | 0.78 | 0.45 |
| | Random | 0.00 | 0.66 | 0.49 |
| **apache math** | AbYSS | 0.00 | 1.70 | 1.12 |
| | GDE3 | 0.87 | 2.10 | 1.11 |
| | MOEAD | 0.93 | 1.70 | 1.17 |
| | NSGAII | 0.86 | 2.20 | 1.16 |
| | OMOPSO | 0.00 | 2.30 | 1.41 |
| | Random | 0.00 | 1.30 | 1.05 |
| **lucene** | AbYSS | 0.00 | 1.50 | 0.83 |
| | GDE3 | 0.56 | 1.20 | 0.78 |
| | MOEAD | 0.56 | 1.70 | 0.90 |
| | NSGAII | 0.56 | 1.30 | 0.82 |
| | OMOPSO | 0.00 | 1.50 | 0.88 |
| | Random | 0.56 | 1.30 | 0.82 |
| **rearchitecturer** | AbYSS | 0.00 | 1.30 | 0.68 |
| | GDE3 | 0.00 | 1.40 | 0.66 |
| | MOEAD | 0.48 | 1.50 | 0.75 |
| | NSGAII | 0.42 | 1.40 | 0.69 |
| | OMOPSO | 0.00 | 1.60 | 0.70 |
| | Random | 0.00 | 0.99 | 0.62 |

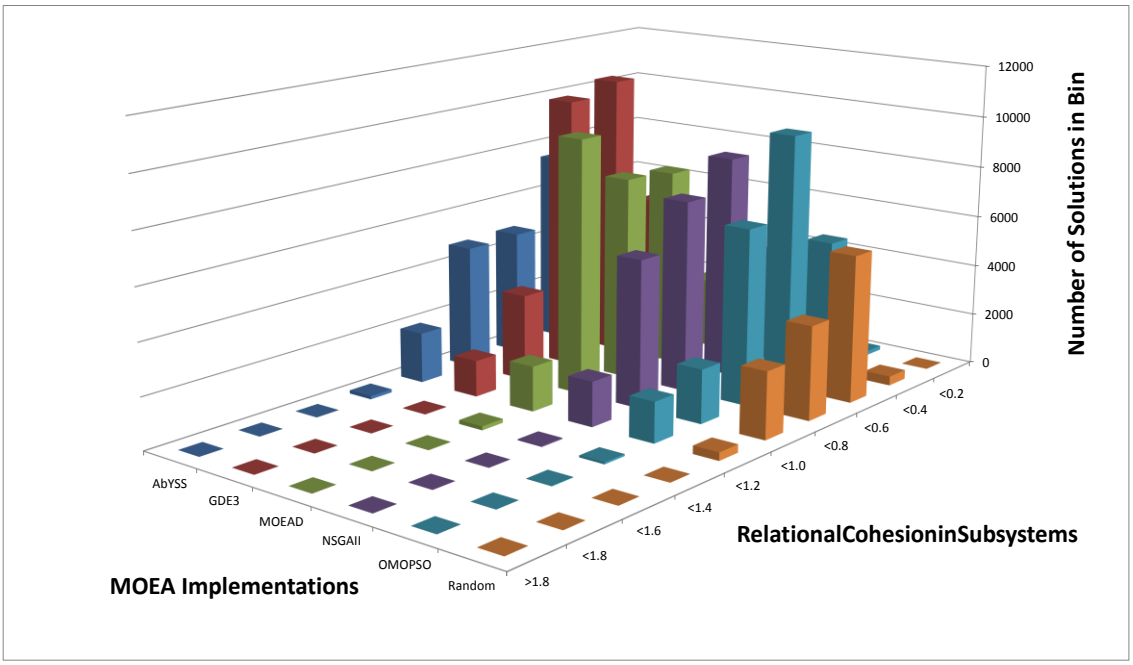


**Figure 48: Histogram *Relational Cohesion in Subsystems* objective**

The descriptive statistics depict that *OMOPSO* outperforms the other *MOEA* implementations for most systems in terms of the absolute achievement in the *Relational Cohesion in Subsystems* metric. The only exception is *MOEAD* with slightly better achievement for the lucene system. However, the histogram shows that only very few solutions with highly promising cohesion characteristics are discovered.

The efferent and afferent coupling metrics feature relatively similar performance characteristics and are presented conjointly. Table 18 and Table 19 present the descriptive statistics of the two objectives.

**Table 18: Descriptive Statistics - *Efferent Coupling of Subsystems***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 46.00 | 250.00 | 152.33 |
| | GDE3 | 30.00 | 240.00 | 137.58 |
| | MOEAD | 31.00 | 230.00 | 132.56 |
| | NSGAII | 47.00 | 250.00 | 132.98 |
| | OMOPSO | 0.33 | 240.00 | 123.95 |
| | Random | 160.00 | 240.00 | 205.16 |
| **apache log4j** | AbYSS | 0.00 | 65.00 | 38.50 |
| | GDE3 | 0.00 | 65.00 | 36.79 |
| | MOEAD | 0.00 | 60.00 | 35.20 |
| | NSGAII | 0.00 | 65.00 | 35.47 |
| | OMOPSO | 0.00 | 65.00 | 36.36 |
| | Random | 0.00 | 65.00 | 46.82 |
| **apache math** | AbYSS | 0.00 | 230.00 | 150.91 |
| | GDE3 | 35.00 | 240.00 | 137.92 |
| | MOEAD | 55.00 | 230.00 | 152.64 |
| | NSGAII | 39.00 | 240.00 | 132.86 |
| | OMOPSO | 0.00 | 230.00 | 108.98 |
| | Random | 0.00 | 220.00 | 194.67 |
| **lucene** | AbYSS | 0.67 | 230.00 | 126.48 |
| | GDE3 | 100.00 | 220.00 | 171.37 |
| | MOEAD | 7.50 | 220.00 | 124.53 |
| | NSGAII | 65.00 | 220.00 | 150.71 |
| | OMOPSO | 0.25 | 220.00 | 119.45 |
| | Random | 75.00 | 220.00 | 153.33 |
| **rearchitecturer** | AbYSS | 0.00 | 100.00 | 65.47 |
| | GDE3 | 0.00 | 100.00 | 58.28 |
| | MOEAD | 20.00 | 90.00 | 60.82 |
| | NSGAII | 1.30 | 100.00 | 57.10 |
| | OMOPSO | 0.00 | 95.00 | 54.19 |
| | Random | 0.00 | 95.00 | 76.68 |

**Table 19: Descriptive Statistics - *Afferent Coupling of Subsystems***

| System | Algorithm | Min | Max | Median |
|---|---|---|---|---|
| **apache ant** | AbYSS | 50.00 | 400.00 | 223.33 |
| | GDE3 | 29.00 | 390.00 | 185.74 |
| | MOEAD | 28.00 | 370.00 | 161.83 |
| | NSGAII | 48.00 | 380.00 | 180.13 |
| | OMOPSO | 0.33 | 380.00 | 178.12 |
| | Random | 250.00 | 380.00 | 311.66 |
| **apache log4j** | AbYSS | 0.00 | 55.00 | 7.22 |
| | GDE3 | 0.00 | 50.00 | 7.42 |
| | MOEAD | 0.00 | 27.00 | 5.18 |
| | NSGAII | 0.00 | 50.00 | 7.45 |
| | OMOPSO | 0.00 | 75.00 | 8.77 |
| | Random | 0.00 | 37.00 | 6.00 |
| **apache math** | AbYSS | 0.00 | 380.00 | 235.93 |
| | GDE3 | 31.00 | 370.00 | 206.81 |
| | MOEAD | 55.00 | 380.00 | 200.84 |
| | NSGAII | 34.00 | 380.00 | 196.04 |
| | OMOPSO | 0.00 | 370.00 | 166.51 |
| | Random | 0.00 | 370.00 | 316.48 |
| **lucene** | AbYSS | 0.00 | 320.00 | 170.74 |
| | GDE3 | 130.00 | 310.00 | 239.18 |
| | MOEAD | 17.00 | 310.00 | 151.40 |
| | NSGAII | 90.00 | 310.00 | 209.65 |
| | OMOPSO | 0.00 | 320.00 | 164.57 |
| | Random | 110.00 | 310.00 | 217.13 |
| **rearchitecturer** | AbYSS | 0.00 | 130.00 | 81.05 |
| | GDE3 | 0.00 | 130.00 | 69.00 |
| | MOEAD | 17.00 | 120.00 | 66.01 |
| | NSGAII | 2.00 | 130.00 | 67.65 |
| | OMOPSO | 0.00 | 120.00 | 66.49 |
| | Random | 0.00 | 120.00 | 98.58 |

The coupling metrics depict a familiar performance profile in which all of the *MOEA* implementations are able to discover promising solutions based on the assessment of the efferent and afferent coupling achievements. However, the maximum coupling values indicate that highly infeasible solutions are also included in the optimal *Pareto-*

*Fronts*. Figure 49 and Figure 50 depict the distributions of the efferent and afferent coupling metrics in the corresponding histograms.

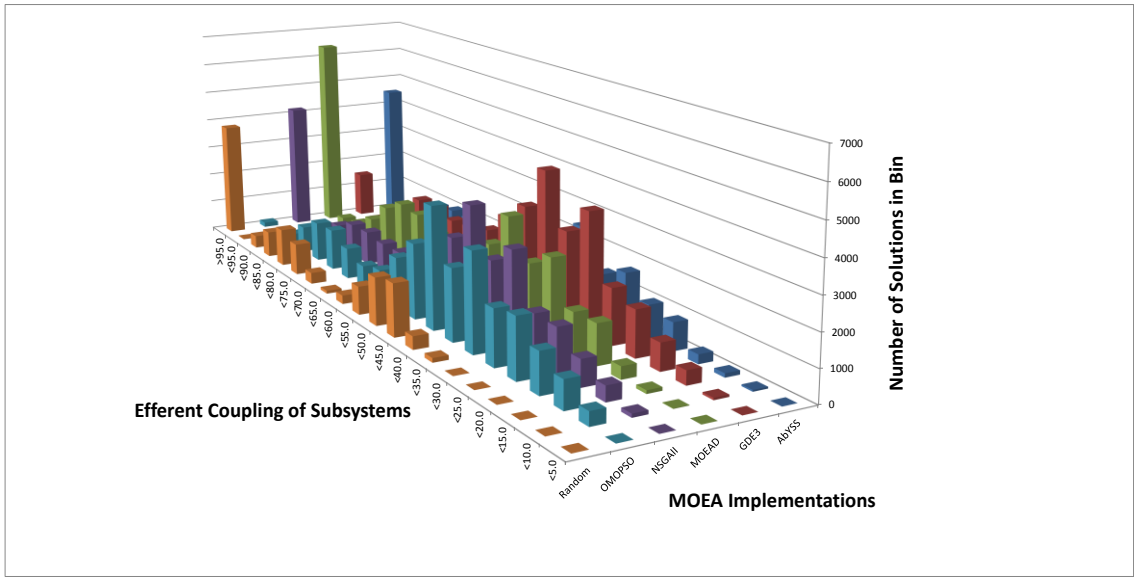


**Figure 49: Histogram *Efferent Coupling of Subsystems* objective**

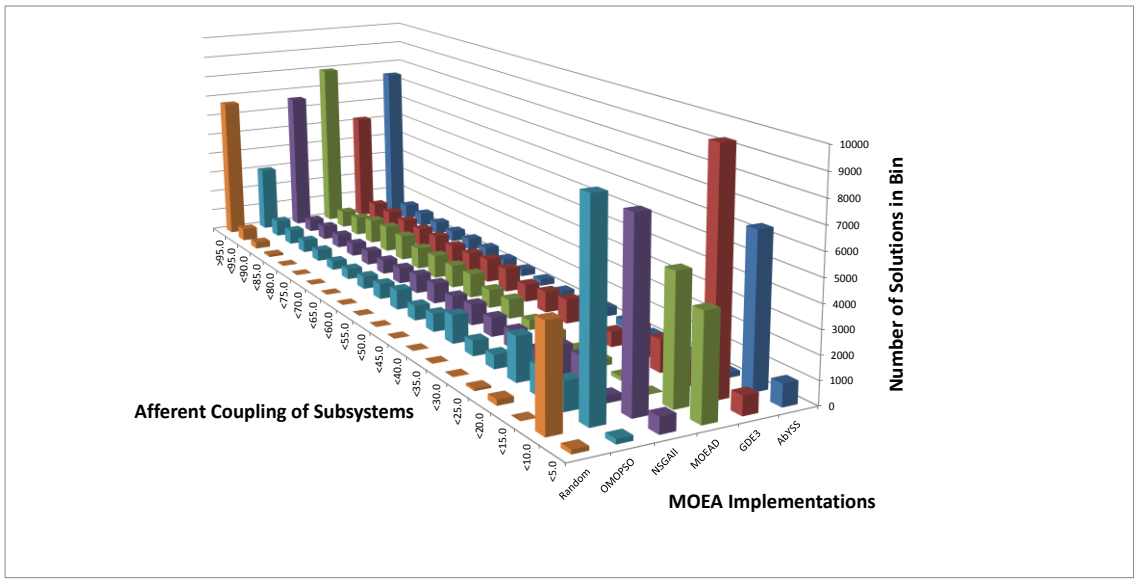


**Figure 50: Histogram *Afferent Coupling of Subsystems* objective**

The histograms of both coupling metrics underpin that the advanced *MOEA* implementations outperform *Random* in both objectives in terms of the depth of the

dive into the promising objective space and the absolute numbers of solutions that feature promising coupling attributes.

The presented tables and histograms show that all of the *MOEA* implementations are able to find acceptable values in the eight objective dimensions. In general, the more advanced *MOEA* implementations feature better performance than the *Random* search implementation. Additionally, the data suggests that *OMOPSO* outperforms the other *MOEA* implementations in the design metrics that are particularly relevant for the construction of high-level architecture designs. This is not only underpinned by the absolute objective achievement but also by the number of solutions within promising objective space over most of the eight objectives. Additionally, the more advanced *MOEA* implementations feature higher numbers of discovered solutions on the optimal *Pareto-Fronts* where *Random* features the lowest numbers on the optimal *Pareto-Fronts*. Hence, the advanced *MOEA* implementations offer stakeholders a higher number of competitive solutions.

It has been discussed in Chapter 4 that stakeholders desire solutions that feature good advancement in the individual objectives as well as good balance between the individual objectives. However, it has been found, based on the presented histograms, that the analysed optimal *Pareto-Fronts* include solutions in almost all objective dimensions that feature unacceptable metric measurements. Hence, the presented data supports the application of constraint techniques over solution sets to reduce the solution sets to a manageable number of highly promising solutions (compare: section 4.2.2).

The presentation of the histograms and descriptive statistics in this section supports the assessment of the performance and distribution characteristics of the *MOEA* implementations in the individual objectives. The implemented evaluation framework supports the calculation of statistical significance tests and effect size measures for each objective based on the generational and optimal *Pareto-Front* solution set and the

descriptive statistics of these solution sets. However, the analysis is very complex as the measures would have to be reported for each objective individually. Additionally, differences in performance would occur in the individual objectives and forming a general conclusion on the performance of the individual configuration slices is very difficult based on the statistical analysis of the individual objectives. Hence, the analysis of the achievement in the individual objective dimensions is not suitable to determine the *overall* performance of the *MOEA* implementations, as only one objective at a time is being been reviewed. A more appropriate method to determine the overall performance is the consideration of multi-objective performance metrics. The next section presents the results of an analysis based on the application of such multi-objective performance metrics.

## 6.4 *Analysis of MOEA performance in the Optimal Pareto-Fronts*

As discussed in section 5.4 *Pareto-Front* performance metrics provide a useful means by which to consolidate the performance of multi-objective metrics into a single comparable value. Multi-objective performance metrics such as *Hypervolume* and *Contribution* have been presented in earlier sections of the thesis to reveal performance differences in multiple variation operator tunings (see: section 5.5) and architecture reconstruction scenarios (see: section 6.2). As presented in section chapter 5, the consideration of multi-objective performance metrics in combination with measures of statistical difference testing are a powerful and straightforward approach to determine and statistically justify the performance differences of multiple search configurations. Hence, the suggested approach is also applied to determine the absolute difference of performance of the employed *MOEA* implementations. Figure 51 depicts the development of the six captured performance metrics based on the discussed dataset and the slicing into the employed *MOEA* implementations.

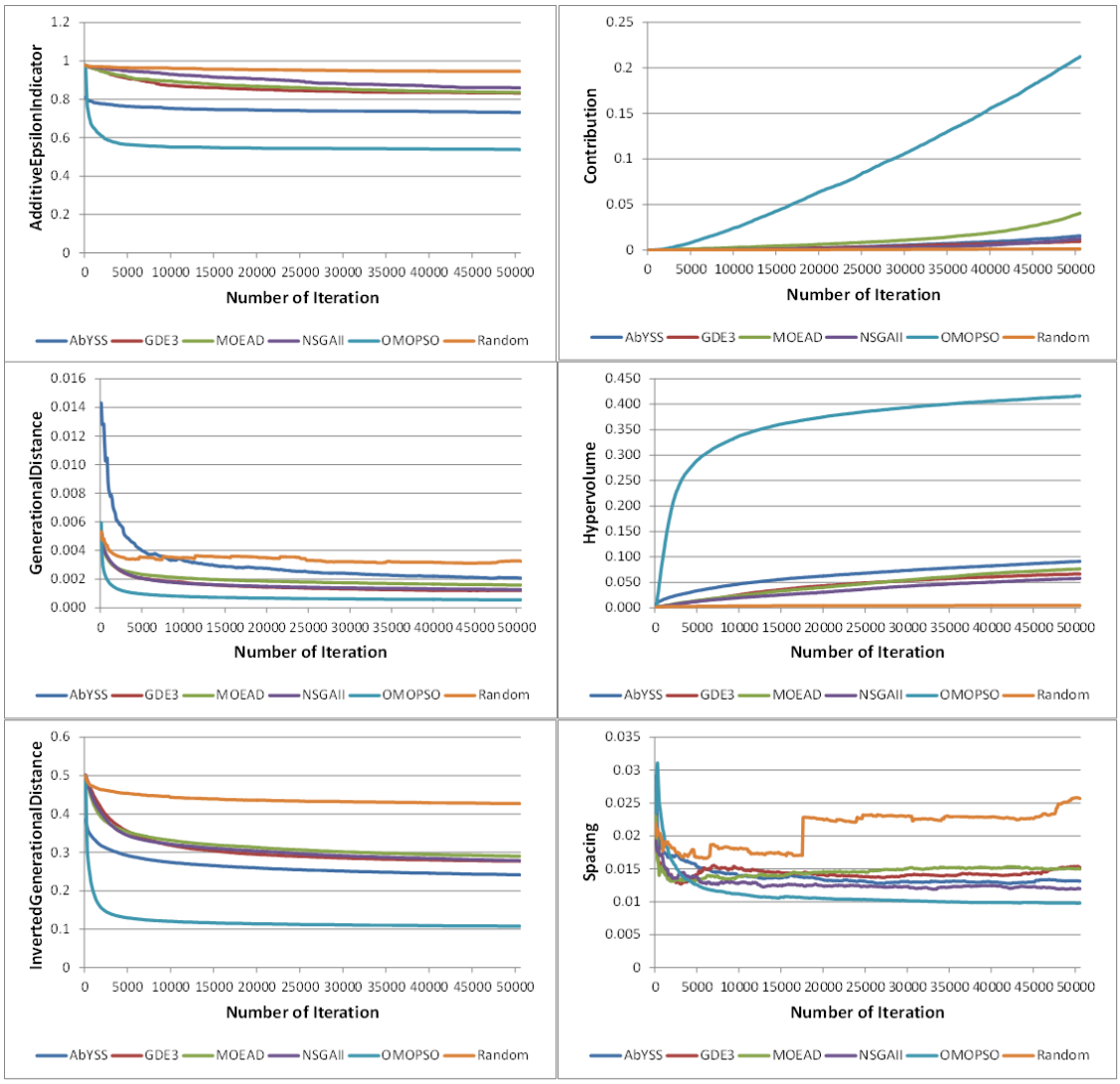


**Figure 51: Performance Indicators - Slicing based on MOEA implementation**

All but the *Spacing* performance metric confirm that all *MOEA* implementations feature better performance than a *Random* search. As discussed in section 3.3.4, it cannot be determined in this research if a wider *Spacing* is a desired *Pareto-Front* attribute in the targeted problem domain. Additionally, the mean development of performance metrics shows that *OMOPSO* outperforms all other *MOEA* implementations in the presented performance indicators. In the performance metrics that measure relative convergence (*Additive Epsilon Indicator*, *Generational Distance,*

*Hypervolume* and *Inverted Generational Distance*) an almost full convergence can be observed between iteration 2,000-3,000.

No major differences in performance development can be observed that depart from the final performance outcome of the *MOEA* implementation slices. Hence, the remaining analysis focuses only on the discussion of the performance snapshot at iteration 50,000. Table 20 depicts the mean performance indicators and the population size of the performance snapshot at iteration 50,000.

**Table 20: Mean Performance of MOEA implementations (Iteration 50,000)**

| *Performance Indicator* | *AbYSS* | *GDE3* | *MOEAD* | *NSGAII* | *OMOPSO* | *Random* |
|---|---|---|---|---|---|---|
| **Spacing** | 0.0131 | 0.0152 | 0.0150 | 0.0120 | 0.0098 | 0.0257 |
| **Inverted Generational Distance** | 0.2621 | 0.3047 | 0.3164 | 0.3075 | 0.1189 | 0.4374 |
| **Hypervolume** | 0.0912 | 0.0666 | 0.0765 | 0.0577 | 0.4162 | 0.0047 |
| **Additive Epsilon Indicator** | 0.7453 | 0.8581 | 0.8690 | 0.8992 | 0.5526 | 0.9538 |
| **Contribution** | 0.0051 | 0.0039 | 0.0115 | 0.0033 | 0.0910 | 0.0007 |
| **Generation Distance** | 0.0021 | 0.0012 | 0.0016 | 0.0013 | 0.0006 | 0.0033 |
| **N** | 180 | 180 | 180 | 180 | 180 | 180 |

The *Kruskal-Wallis* test has been applied due to the departure from normality in all populations in the performance snapshot at iteration 50,000. The pair-wise significance comparisons of the *MOEA* implementations with *Random* search, and of *OMOPSO* with any other *MOEA* implementation, confirmed a statistically significant difference in all performance metrics. The pair-wise comparison of *NSGAII*, *GDE3*, AbYSS and *MOEAD* in the *Additive Epsilon Indicator*, *Contribution*, *Generational Distance*, *Hypervolume*, and *Inverted Generational Distance* performance indicator measures do not feature any noteworthy statistically significant outcomes. (Note that presentation of the p-values has limited value in terms of determining the actual performance difference of the individual *MOEA* implementations. Hence, the reporting of the p-values is omitted here.

Nevertheless, the corresponding datasets can be downloaded from the project webpage[42].)

It is anticipated that the reporting of the effect size measures across the individual performance metrics offers more value to those assessing the performance differences of the individual *MOEA* implementations. The evaluation framework calculates the effect size for any combination of *MOEA* implementations. However, the presentation of these results would be likely overwhelming. Instead, the presentation of the comparison of the *MOEA* implementations with a *Random* search is useful and enables an effective performance comparison of the employed *MOEA* implementations. Correspondingly, Table 21 presented the effect size measures of the calculated multi-objective performance metric indicators.

**Table 21: *Cohens'd* Effect-Size of Performance Indicators (Iteration 50,000)**

| | **AbYSS vs. Random** | **GDE3vs. Random** | **MOEAD vs. Random** | **NSGAII vs. Random** | **OMOPSO vs. Random** |
|---|---|---|---|---|---|
| **AEI** | 3.50 | 1.44 | 1.37 | 1.06 | 3.96 |
| **Contribution** | 0.99 | 1.09 | 1.18 | 0.98 | 2.68 |
| **GD** | 0.21 | 1.06 | 0.78 | 1.00 | 1.48 |
| **Hypervolume** | 1.74 | 1.52 | 1.27 | 1.40 | 3.13 |
| **IGD** | 1.58 | 1.28 | 1.05 | 1.22 | 3.43 |
| **Spacing** | 0.32 | 0.26 | 0.27 | 0.36 | 0.42 |

The effect size table confirms that the advanced *MOEA* implementations feature a large effect size in relation to the *Random* search implementation in the three most established performance indicator measures (*Additive Epsilon Indicator*, *Hypervolume, Inverted Generational Distance*). Additionally, *OMOPSO* features the overall best performance against *Random* in these performance measures with an effect size value of more than 3.0. *AbYSS*, *GDE3*, *MOEAD* and *NSGAII* feature relatively similar

[42] http://code.google.com/p/rearchitecturer/wiki/evaluationResults

performance in all of the performance indicators, with slightly better performance of *AbYSS* in the *Additive Epsilon Indicator*, *Hypervolume, and Inverted Generational Distance* performance metric. Exception are the observations of the *Generational Distance* and *Contribution* metric in which *AbYSS* has a relatively low performance in comparison with its immediate competitors *GDE3*, *MOEAD* and *NSGAII*. Additionally, *OMOPSO* features only a *Generational Distance* effect size of 1.48 and a slightly lower contribution measure of 2.68. However, these observations align with the general criticism of the *Generational Distance* and *Contribution* performance metrics (compare: section 3.3.4).

## 6.5 *Summary*

This chapter presented the results of the application of the *Rearchitecturer* in an architecture reconstruction scenario with eight objectives. The architecture configuration of five open-source software systems was transformed towards three different target architecture models. It was found that the employment of different variation operator settings has only a marginal impact on the search performance. Additionally, the employment of different target architecture models showed that only the upstream mounting of packages into high-level artefacts impacts the search negatively. However, only a small effect size could be found. Hence, the evaluation confirmed that the developed approach is applicable across different architecture reconstruction scenarios and features relatively similar performance results in different architecture reconstruction scenarios.

Furthermore, the generational population advancement of the employed *MOEA* implementations in the individual objectives was examined. It was shown that the approach is able to converge the employed high-level architecture metrics towards the desired optimisation target. Additionally, distribution characteristics of the achieved optimal *Pareto-Front* were discussed. It was found that the system size impacts the search performance. Furthermore, the performance of the individual *MOEA* implementation was analysed based on the review of the achievement in multi-

objective performance indicator metrics. It was shown that the advanced *MOEA* implementations feature better performance than the *Random* search implementation. Furthermore, in particular the *OMOPSO* implementation that employs a relaxed form of *Pareto-Dominance* featured the overall best performance in the evaluated application scenarios. Additionally, *OMOPSO* was the only *MOEA* implementation that was able to achieve acceptable objective values in the architecture metrics in the more complex software systems.

# 7 Conclusion

This chapter concludes the research presented in this thesis. Firstly, it discusses the contributions made in the area of search based software architecture reconstruction. Secondly, limitations of this research are summarised and finally areas of further research are discussed.

## 7.1 *Discussion of Contributions*

The main objective of this research has been to evaluate the feasibility of multi-objective optimisation strategies when applied in the application domain of architecture reconstruction. This objective has been successfully achieved by the implementation of the *Rearchitecturer* tool that has been used to evaluate different optimisation strategies and algorithms applied to architecture reconstruction. This has provided insights into the process as well as strong contributions to the body of knowledge in this domain. The following sections discuss the main contributions and findings of this research.

### *7.1.1 A Multi-Objective Architecture Reconstruction Framework*

The review of related literature reported primarily in Chapter 2 revealed that no research has been conducted that assesses the performance of different *MOEA* implementations in the application domain of software modularisation, and the specific applicability of high-level architecture metrics to reconstruct architecture designs. This research addressed these gaps by contributing a novel architecture reconstruction framework that enables the flexible employment of *MOEA* configurations and high-level architecture design metrics. The framework is manifested in a substantial open-source software system, called *Rearchitecturer,* that classifies low-level software artefacts into conceptual high-level target architecture models.

Previous research in the domain of search based software modularisation, as exemplified in Mitchell (2002), Seng et al. (2005), Abdeen et al. (2009) and Praditwong et al. (2011), focuses in particular on the grouping of low-level artefacts (classes/files) into the next higher abstraction level (packages/folders). The new framework

implemented in this research also enables the operation of classifications on a higher abstraction level (classification of packages into subsystems). The framework thus allows the classification of compilation units into packages and/or the classification of packages into the subsystems of a conceptual architecture model. Furthermore, the approach enables the redevelopment of the entities and dependencies of conceptual architecture artefacts (packages, subsystems and dependencies between subsystems). Constraints can be defined to limit the number of conceptual artefacts within the architecture e.g. maximum number of packages and/or maximum number of subsystems in the architecture model.

Within previous research the artefacts of the abstraction level are developed during the modularisation process and the approaches do not allow inducing high level artefact configurations. The principle of software architecture design that the conceptual architecture model describes the purpose of the software system instead of being a reflexion of the implementation of the design, as emphasized in Fowler (2002) and Martin (2011), is not supported by such approaches. The present research therefore contributes an approach that separates the employment of the classification of low-level artefacts and the development of the high-level structure. The conceptual architecture model can be induced into the classification process. As such, this conceptual architecture model can operate as a blueprint for the aspired design of the system.

The design of high-level architectures depends on multiple factors, such as the desired modularity, the module dependencies, the application domain and implemented high-level architecture frameworks. The developed architecture reconstruction framework gives stakeholders flexible control over the configuration of the employed objectives to express such different requirements. A variety of established high-level architecture design metrics are implemented in the *Rearchitecturer* prototype and can be employed as search objectives. This flexible

control of the objective configuration is a distinct feature in comparison to other *SBSE* approaches in which the objective configuration is pre-defined.

In the presented approach, optimisation configurations can be created that feature many objectives. The application of optimisation approaches with a high number of objectives is complex. *Pareto-Fronts* with more than three dimensions cannot be effectively visualised and the application of multi-objective problems that feature more than three objectives usually results in optimal *Pareto-Fronts* with many solutions (He & Yen, 2014). The high number of solutions in such optimal *Pareto-Fronts* quickly exceeds the abilities of stakeholders to manually review the individual solutions. It has been discussed that stakeholders most likely desire only one optimal solution that can be applied in the architecture management process. To this date no approach had been suggested that supports stakeholders in identifying the most promising solution of the optimal *Pareto-Front*. This research contributes a novel candidate approach that supports stakeholders to iteratively refine the space of acceptable solution attributes across all collected architectural metrics to efficiently reduce the number of solutions in the reviewed solution set. Hence, architectural metrics can be constrained even if they were not utilised as search objectives. The identified highly promising solutions can be graphically visualised to support stakeholders in identifying a final solution that can be employed in the architecture management process.

In conclusion, the contribution of the implemented dynamic problem representation in the developed framework enables the reconstruction of software architecture configurations on different abstraction levels and considers aspects of the desired conceptual architecture model. This dynamic problem representation in combination with the implemented flexible objective configuration approach contribute a framework that can help the user to gain valuable insight into the problem domain of architecture reconstruction and software modularisation. Additionally, the extensible integration of a range of established optimisation libraries now enables the application and

performance evaluation of a previously inaccessible diversity of *MOEA* implementations and variation operator tunings in architecture reconstruction application contexts.

### *7.1.2 A Multi-Objective Evaluation Framework*

The design of the *Rearchitecturer* prototype enables the employment of a variety of *MOEA* implementations with different objective configuration, architecture reconstruction and variation operator settings. A representative evaluation of the objectives of this research demands the employment of multiple *MOEA* implementations in different software systems, architecture configurations and *MOEA* tunings. Due to the probability-based characteristics of the applied *MOEA* implementations these different evaluation scenarios create optimisation configurations that need to be executed multiple times. Descriptive and inductive statistical measures need to be calculated to identify different effects between optimisation configurations. The review of applicable evaluation frameworks in the area of multi-objective evaluation approaches revealed that no generally established process exists to empirically evaluate and compare the results of multi-objective optimisation approaches. As there is no such process, researchers often apply different approaches or use different measures to evaluate performance. This makes it challenging for researchers to gain useful and conclusive insight from existing results.

This research therefore contributes a novel multi-objective evaluation framework that analyses optimisation results independently of the problem-formulation. The framework enables the slicing and agglomeration of optimisation result sets based on user definition of configuration parameters to evaluate the impact of configuration settings on the search performance. The implemented evaluation framework enables the user to analyse achievement in the objective space for the set of visited solutions and the set of optimal *Pareto-Front* solutions. Histograms of objective achievement can be created to enable the review of the distribution characteristics of such solutions sets in the objective space. Additionally, the evaluation framework integrates a set of six multi-objective performance metrics to support user assessment of the composition and

convergence of optimal *Pareto-Fronts.* Descriptive statistics are calculated for the objective achievement as well as multi-objective *Pareto-Front* metrics based on the configured data-set slicing. Methods of statistical difference testing are employed corresponding to the normality characteristics of the data slices. Additionally, effect size measures are calculated to determine an absolute measure of difference between the individual data slices. In conclusion, the novel evaluation framework contributes a structured means to evaluate the performance of multi-objective approaches independently of the actual problem formulation. The configurable agglomeration and slicing approach of optimisation datasets based on configuration attributes contributes a powerful method to evaluate the impact of specific configuration parameter tunings on the search performance in more general optimisation configuration contexts. In addition, it has the potential to be applied across a wide range of different application domains to ensure consistency and comparability of results.

### *7.1.3 Experimental Evaluation of Architecture Reconstruction Approach*

The performance of six *MOEA* implementations (*MOEAD*, *GDE3*, *NSGAII*, *AbYSS*, *OMOPSO* and *RandomSearch)* has been evaluated in the application domain of architecture reconstruction. The applied *MOEA* implementations have been selected based on their different search strategies and wide application in other *SBSE* research (*NSGAII - Genetic Algorithm*, *AbYSS – Scatter Search*, *GDE3 - Differential Evolution*). Additionally, *MOEA* implementations are employed that feature search strategies that have been found to overcome the challenges in complex optimisation configurations with many objectives. These search strategies are a decomposition-based *MOEA* implementation (*MOEAD*) and a relaxed form of *Pareto-Dominance* (ε-dominance) incorporated in a particle swarm implementation (*OMOPSO*). *Random* search has been used as a benchmark to determine if the employed advanced *MOEA* implementations feature better performance than an approach that relies on a random creation of solutions.

A reconstruction scenario is employed that assigns compilation units into packages and packages into the high-level subsystems of a pre-defined target architecture model. Three different target architecture models are employed to evaluate the performance of the *MOEA* implementation in reconstruction scenarios that feature different architectural requirements. The six *MOEA*s are employed to transform the architectural composition of five open-source software systems into the three target architecture model designs. The five open-source software systems vary in size (compare: section 3.3.3) to provide insights into the performance of the *MOEA* implementations in systems that feature different levels of complexity.

In this experimental evaluation eight high-level software design metrics are employed as search objectives. This objective configuration features cohesion and coupling metrics similar to configurations that have been employed in related research, but also features specific high-level architecture design metrics such as the *Number of Cycles in Packages*, *Number of Forbidden Type Dependencies*, *NCCD* and *Distance from Main Sequence* that have not been previously applied in a multi-objective modularisation approach. The analysis showed that the application of such high-level architecture metrics is feasible and that the *MOEA* implementations are capable of driving the search towards solutions that feature highly promising solution attributes (compare: section 6.3). Nevertheless, the evaluation also showed that some *MOEA* implementations (*AbYSS*, *GDE3*, *NSGAII, MOEAD*) could only achieve highly promising solutions for the less complex software systems. An exception has been the *OMOPSO* implementation that also discovered some highly promising solution objective measures in all of the transformed software systems. It has also been observed that the other *MOEA* implementations (*AbYSS, GDE3, MOEAD, NSGAII*) are able to discover a higher number of solutions than *OMOPSO* in the relatively promising objective space. This observation has been made in the *Cohesion in Subsystems, Distance from Main Sequence, Number of Cycles* and *Range of Types in Subsystems* objectives.

Furthermore, the performance of the optimal *Pareto-Front* achievement of the *MOEA* implementations has been reviewed based on the application of multi-objective performance indicators (compare Figure 51). It has been found that all advanced *MOEA* implementations feature significantly better performance than *Random* for the *Additive Epsilon Indicator, Contribution, Generational Distance, Hypervolume and Inverted Generational Distance* performance indicators. These performance indicators also confirm the superior performance of the *OMOPSO* implementation. These performance indicators show relatively similar performance results for the remaining *MOEA* implementations. Based on the overall analysis of the performance indicator results, *OMOPSO* features the overall best *Pareto-Front* and objective achievement in the scenarios considered here.

It is likely, however, that the prominent performance of the *OMOPSO* implementation is mainly based upon the application of the relaxed form of *Pareto-Dominance.* Research that analysed the performance of standard *MOPSO* and *NSGAII* implementations found relatively poor performance of these *MOEA* implementations in objective formulations with many objectives (Köppen & Yoshida, 2007; Mostaghim & Schmeck, 2008). The strict concept of *Pareto-Dominance* is not fully suited to the application of many objectives. *Pareto-Dominance* requires a better objective performance in at least one objective and equal performance in the remaining vectors to include a solution in the *Pareto-Front*. This is more and more unlikely if the number of objectives increases. Hence, the *MOPSO* implementation may get trapped by the local best selection of the particles itself without being able to find new dominating positions. The application of ε-dominance as a relaxed form of *Pareto-Dominance* as suggested in Sierra and Coello (2005) proved to be a prominent strategy to overcome the challenges of many objectives in the application domain of software modularisation.

Another strategy that is potentially promising to overcome the challenges of configurations with many objectives is the application of decomposition based *MOEA* implementations (He & Yen, 2014). However, the performance indicator measures of

the applied *MOEAD* implementation did not provide any particular evidence of a better performance of the *MOEAD* implementation in comparison to the *AbYSS*, *NSGAII* and *GDE3* implementations.

Furthermore, the impact of different variation operator settings and architecture reconstruction scenarios on the search performance has been evaluated. The evaluation of the impact of different variation operator tunings revealed significant differences between tunings. However, the effect size measures showed no mentionable performance effects between the individual tunings. Hence, the tuning of the variation operator parameters has only a marginal impact on the performance of the search in the developed problem formulation (compare: section 6.1). The application of different target architecture models revealed the existence of a slight effect on the search performance based on the complexity on the applied target architecture model (compare: section 6.2).

In conclusion, the experimental results discussed in this section contributed empirical evidence supporting the applicability and performance of evolutionary algorithms, beyond the application of basic *GA* implementations, in the application domain of software architecture reconstruction. Furthermore, the application of the developed multi-objective evaluation framework contributes evidence that a tool-supported process for the evaluation of the performance of optimisation configuration settings is feasible.

## 7.2 *Limitations*

This research has been a significant but still initial effort to assess the applicability of multi-objective search driven techniques to transform the architectural configuration of software systems towards desired architectural designs. The initial and explorative character of this research implies some limitations that are discussed in this section.

The present research focuses on the reconstruction of architecture models of software systems by applying software quality metrics as fitness functions. In theory an architecture model should describe what a system is supposed to do and not be a representation of the system's implementation. This work partially addressed this by allowing the inclusion of user-defined layering and subsystems paired with dependency configurations. These configurations can and should represent technical and/or domain aspects of the system. However, the classification into these modules is based on the fitness of software architecture metrics. Hence, it is acknowledged that a modularisation approach that is driven by optimising architecture design metrics, as suggested in this research, is not capable of extracting domain aspects of the source base and assigning these into the corresponding artefacts of the conceptual architecture model. Nevertheless, the assumption within this research is that such modularisation approaches are capable of finding partitionings that fit the targeted conceptual architecture model from a software design perspective, and that these configurations can operate as a feasible basis for further development and architecture compliance checking. The complementation with approaches to identify and classify domain and/or technical aspects remains a challenge to create applicable architecture reconstruction approaches.

The presented approach is able to assign compilation units to packages and to assign existing or new packages to high-level artefacts. The assignment of compilation units into packages does not change the design of the system on the micro-design level. Low-level refactoring such as create compilation unit, move member and split method would be required to change the design on the micro-level to change the dependency structure of the physical architecture on the compilation unit level. Initial experiments with low-level refactorings have been conducted in this research. However, insufficient performance results have been achieved which prevented a rigorous experimental analysis within a multi-objective optimisation approach. Nevertheless, the problem that

such refactorings would be driven by software engineering metrics and would not include domain aspects would remain for the application of low-level refactorings.

Furthermore, the designed approach supports the employment of a wide set of different optimisation goals, conceptual architecture patterns, *MOEA* implementations and tunings. In this research only one objective setting, a set of six *MOEA* implementations, a set of three architecture patterns, one mutation operator and crossover operator have been evaluated. This is certainly not a complete exploration of the available configuration aspects. Nevertheless, it has been attempted to select these elements in a rigorous way based on experimental and literature review findings. However, there is a chance that other optimisation configurations exist that could contribute other valuable insights that were not gained in the employed evaluation.

As a proof of concept, the architecture configurations of five open-source software systems have been transformed in the evaluation of this research. The selected systems feature a different number of packages, number of compilation units and number of dependencies to accommodate the conduct of a representative evaluation. The most recent source code releases, at the time of the conduct of the evaluation, have been used to enable an evaluation that is as current as possible. Nevertheless, the reliance on such a set of software systems implies that only reliable statements on the performance of the approach in other systems can be made if these systems were to feature similar structural attributes as the original evaluation systems.

## 7.3 *Future Work*

This section discusses the areas of potential further research that have emerged during this research project.

This research relied on the validation of the proposed approach based on quantitative measures of quality and performance. More broadly, the development of the architecture reconstruction prototype has been driven by its potential application in

a software development context. The consideration of relevant context workflows is based on the experience of the researcher in the area of software architecture management and monitoring. However, these workflows are unverified in terms of the potential integration of a multi-objective architecture reconstruction approach in the software development process. Their application in real software engineering contexts with involvement of development stakeholders contributing explicit system knowledge would be useful to assess the genuine applicability of the developed approach.

A potential problem of such an end-user evaluation is that the current design of the tool exposes the end-user heavily to the concepts and tuning of MOEAs. Hence, at this stage of the tool's development a good understanding of MOEAs and their tuning is essential. Further development based on the insights gained in this research could hide the MOEA tuning from the end-user and recommend settings that have delivered good convergence in previous optimisations. Nevertheless, the general concepts of MOEAs (such as optimal Pareto-Fronts) need to be understood for the effective use of the tool. A potential scenario for the end-user evaluation would be the application of software architecture engineers (e.g. refactoring consultants) who are educated in the *MOEA* concepts. These software architecture engineers could then advise development stakeholders on potential refactoring strategies based on the insights gained in the application of the tool.

In this research the employed objective setting features a total of eight objectives. One of the reasons for the employment of such a high number of objectives has been to demonstrate the feasibility of the approach to deal with quite complex objective configurations. Nevertheless, the application of so many objectives increases the complexity of the search and renders some *MOEA* algorithms as infeasible. It is worth investigating if these *MOEA* implementations deliver better achievement outcomes in objective configurations with fewer objectives.

Additionally, it has been discussed that other techniques exist to enable *MOEA* implementations to overcome the challenges of complex objective configurations (grid-based and performance-indicator based *MOEA* implementations). However, at this stage the *Rearchitecturer* component does not support *MOEA* implementations that utilise such strategies.

Furthermore, the results presented in the evaluation of this thesis have often been consolidated to enable statement to be made regarding the general feasibility of the developed approach. Such an analysis carries the risk that interesting insights are obfuscated by the reliance on average (or similar) values. A more detailed analysis might provide valuable insights into certain specific performance differences (e.g. analysis of the impact of a system's size on the search performance, or analysis of the impact of different variation operator tunings in different *MOEA* implementations).

Finally, the implemented evaluation framework should also be useful in future research that addresses such ongoing questions in the problem area of *MOEA*-driven architecture reconstruction (e.g. application of additional *MOEA* implementations, different objective configurations, *MOEA* tunings and reconstruction configurations).

## 7.4 *Conclusion*

The main objective of this work has been to evaluate the feasibility of multi-objective optimisation strategies when applied in the area of architecture reconstruction, to identify feasible architecture classifications that can operate as a starting configuration for the modularisation of software systems and the containment of software erosion.

It has been shown in this work that it is indeed feasible to find software architecture configurations by applying multi-objective optimisation techniques. It has been demonstrated in the evaluation that the search converges the architecture configurations towards desired software architecture design metrics. Solutions that

feature objective measures, that would be acceptable in practise, in all objectives of the applied objective configuration, could be identified within the smaller software systems that have been transformed in this study.

The inclusion of conceptual target architecture models in combination with the number of *Forbidden Type Dependency*, *Number of Cycle* and *NCCD* metrics enables the identification of modular software architecture configurations that align with the desired high-level design. Furthermore, the identified software architecture configurations can be utilised in the future software architecture management and monitoring process to identify the occurrence of new violations that might occur during ongoing development of the system. Hence, the created architecture configuration can be a first stepping stone to contain the further erosion of a software system.

# 8 References


Abdeen, H., Ducasse, S., Sahraoui, H., & Alloui, I. (2009). Automatic package coupling and cycle minimization. *Proceedings of 16th Working Conference on Reverse Engineering (WCRE'09)*, 103-112.

Al-Mutawa, H. A., Dietrich, J., Marsland, S., & McCartin, C. (2014). On the Shape of Circular Dependencies in Java Programs*IEEE.* Symposium conducted at the meeting of the Software Engineering Conference (ASWEC), 2014 23rd Australian

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., & Meedeniya, I. (2013). Software architecture optimization methods: A systematic literature review. *IEEE Transactions on Software Engineering, 39*(5), 658-683. doi:10.1109/TSE.2012.64

Altmann, D. G. (1991). *Practical statistics for medical research*. London, UK: Chapman and Hall.

Amoui, M., Mirarab, S., Ansari, S., & Lucas, C. (2006). A genetic algorithm approach to design evolution using design pattern transformation. *International Journal of Information Technology and Intelligent Computing, 1*(2), 235-244.

Andrews, P. S. (2006). An investigation into mutation operators for particle swarm optimization. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 1044-1051. doi:10.1109/CEC.2006

Anquetil, N., & Lethbridge, T. C. (1999). Experiments with clustering as a software remodularization method. *Proceedings of the Sixth Working Conference on Reverse Engineering (WCRE)*, 235-255.

Antunes, P., Zurita, G., & Baloian, N. (2014). An application framework for developing collaborative handheld decision-making tools. *Behaviour & Information Technology, 33*(5), 470-485. doi:10.1080/0144929X.2013.815275

Arcuri, A., & Briand, L. (2011). A practical guide for using statistical tests to assess randomized algorithms in software engineering. *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, 1-10. doi:10.1145/1985793.1985795

Assunção, W. K. G., Colanzi, T. E., Pozo, A. T. R., & Vergilio, S. R. (2011). Establishing integration test orders of classes with several coupling measures. *Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation (GECCO)*, 1867-1874. doi:10.1145/2001576.2001827

Aversano, L., Canfora, G., Cerulo, L., Del Grosso, C., & Di Penta, M. (2007). An empirical study on the evolution of design patterns. *Proceedings of the the 6th joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT (Symposium on the Foundations of Software Engineering)*, 385-394. doi:10.1145/1287624.1287680

Bagnall, A. J., Rayward-Smith, V. J., & Whittley, I. M. (2001). The next release problem. *Information and Software Technology, 43*(14), 883-890. doi:10.1016/S0950-5849(01)00194-X

Barros, M. d. O. (2012). An analysis of the effects of composite objectives in multiobjective software module clustering. *Proceedings of the 14th annual*

*Conference on Genetic and Evolutionary Computation Conference (GECCO)*, 1205-1212. doi:10.1145/2330163.2330330

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Reading, MA: Addison-Wesley Professional.

Beck, K. (2003). *Test-driven development: By example*. Reading, MA: Addison-Wesley Professional.

Blasco, X., Herrero, J., Sanchis, J., & Martínez, M. (2008). A new graphical visualization of n-dimensional Pareto front for decision-making in multiobjective optimization. *Information Sciences, 178*(20), 3908-3924. doi:10.1016/j.ins.2008.06.010

Bosch, J. (2010). Architecture in the age of compositionality. *Proceedings of the 4th European Conference (ECSA) Copenhagen, Denmark*, 1-4. doi:10.1007/978-3-642-15114-9_1

Brasil, M. M. A., da Silva, T. G. N., de Freitas, F. G., de Souza, J. T., & Cortés, M. I. (2012). A multiobjective optimization approach to the software release planning with undefined number of releases and interdependent requirements. *Proceedings of the 13th International Conference of Enterprise Information Systems (ICEIS)*, 300-314. doi:10.1007/978-3-642-29958-2_20

Breivold, H. P., Crnkovic, I., & Larsson, M. (2012). A systematic review of software architecture evolution research. *Information and Software Technology, 54*(1), 16-40. doi:10.1016/j.infsof.2011.06.002

Burgess, C. J., & Lefley, M. (2001). Can genetic programming improve software effort estimation? A comparative evaluation. *Information and Software Technology, 43*(14), 863-873. doi:10.1016/S0950-5849(01)00192-6

Coello, C. A. C., Lamont, G. B., & Van Veldhuisen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems*. New York, NY: Springer.

Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. Hillsdale, NJ: Routledge Academic.

Connor, A. M., & Shah, A. (2014). Resource allocation using metaheuristic search. *Proceedings of the 4th International Conference on Computer Science and Information Technology*, 353 - 364. doi:10.5121/csit.2014.4230

Dalgarno, M. (2009). When good architecture goes bad. *Methods & Tools, 17*, 27-34.

De Silva, L., & Balasubramaniam, D. (2012). Controlling software architecture erosion: A survey. *Journal of Systems and Software, 85*(1), 132-151. doi:10.1016/j.jss.2011.07.036

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*. New York, NY: John Wiley & Sons Chichester.

Deb, K., & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems, 9*, 115 -148.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture notes in computer science, 1917/2000*, 849-858. doi:10.1007/3-540-45356-3_83

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions, 6*(2), 182-197. doi:10.1109/4235.996017

Di Penta, M., Cerulo, L., Guéhéneuc, Y.-G., & Antoniol, G. (2008). An empirical study of the relationships between design pattern roles and class change proneness. *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, 217-226.

Dijkstra, E. W. (1972). The humble programmer. *Communications of the ACM, 15*(10), 859-866. doi:10.1145/355604.361591

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association, 56*(293), 52-64. doi:10.1080/01621459.1961.10482090

Durillo, J. J., Nebro, A. J., Luna, F., & Alba, E. (2008). Solving three-objective optimization problems using a new hybrid cellular genetic algorithm. *Proceedings of the Conference on Parallel Problem Solving from Nature - PPSN X*, 661-670. doi:10.1007/978-3-540-87700-4_66

Edgeworth, F. Y. (1881). *Mathematical psychics: An essay on the application of mathematics to the moral sciences*. London, UK: Kegan Paul.

Eiben, A. E., Michalewicz, Z., Schoenauer, M., & Smith, J. E. (2007). Parameter control in evolutionary algorithms. In F. G. Lobo, C. F. Lima, & Z. Michalewicz (Eds.), *Parameter setting in evolutionary algorithms* (pp. 19-46). Berlin, DE: Springer.

Eiben, A. E., & Smith, J. E. (2003). Introduction. In A. E. Eiben & J. E. Smith (Eds.), *Introduction to evolutionary computing* (pp. 1-14). Berlin, DE: Springer.

Eick, S. G., Graves, T. L., Karr, A. F., Marron, J. S., & Mockus, A. (2001). Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering, 27*(1), 1-12. doi:10.1109/32.895984

Etemaadi, R., & Chaudron, M. R. (2012). Varying topology of component-based system architectures using metaheuristic optimization. *Proceedings of the Conference on Software Engineering and Advanced Applications (SEAA)*, 63-70.

Etemaadi, R., Emmerich, M. T., & Chaudron, M. R. (2012). Problem-specific search operators for metaheuristic software architecture design. In G. Fraser & J. Teixeira de Souza (Eds.), *Search Based Software Engineering: Proceedings of the 4th International Symposium, SSBSE 2012, Riva del Garda, Italy, September 28-30, 2012.* (pp. 267-272). Berlin, DE: Springer.

Falleri, J.-R., Denier, S., Laval, J., Vismara, P., & Ducasse, S. (2011). Efficient retrieval and ranking of undesired package cycles in large software systems. In *Objects, Models, Components, Patterns* (pp. 260-275): Springer.

Fenton, E. N., & Neil, M. (2000). Software metrics: Roadmap. *Proceedings of the Conference on The Future of Software Engineering (ICSE '00)*, 357-370 doi:10.1145/336512.336588

Ferrer, J., Chicano, F., & Alba, E. (2012). Evolutionary algorithms for the multi-objective test data generation problem. *Software: Practice and Experience, 42*(11), 1331-1362. doi:10.1002/spe.1135

Forward, A., & Lethbridge, T. C. (2002). The relevance of software documentation, tools and technologies: A survey. *Proceedings of the 2002 ACM Symposium on Document Engineering (DocEng '02)*, 26-33. doi:10.1145/585058.585065

Fowler, M. (1999). *Refactoring: Improving the design of existing code*. Boston, MA, USA: Addison-Wesley Professional.

Fowler, M. (2001). Reducing coupling. *IEEE Software, 18*(4), 102-104. doi:10.1109/MS.2001.936226

Fowler, M. (2002). *Patterns of enterprise application architecture*. Boston, MA: Addison-Wesley.

Freeman, E. (2004). *Head first design patterns*. Sebastopol, CA: O'Reilly Media.

Frey, S., Fittkau, F., & Hasselbring, W. (2013). Search-based genetic optimization for deployment and reconfiguration of software in the cloud. *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 512-521.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading, MA: Addison-Wesley.

Glover, & Kochenberger, G. A. (2003). *Handbook of metaheuristics*. Berlin, DE: Springer.

Godfrey, M. W., & Lee, E. H. (2000). Secrets from the monster: Extracting Mozilla's software architecture. *Proceedings of Second Symposium on Constructing Software Engineering Tools (CoSET'00), Limerick, Ireland*, 15-23.

Greer, D., & Ruhe, G. (2004). Software release planning: An evolutionary and iterative approach. *Information and Software Technology, 46*(4), 243-253. doi:10.1016/j.infsof.2003.07.002

Gui, G., & Scott, P. D. (2006). Coupling and cohesion measures for evaluation of component reusability. *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR)*, 18-21. doi:10.1145/1137983.1137989

Hamdan, M. (2011). A dynamic polynomial mutation for evolutionary multi-objective optimization algorithms. *International Journal on Artificial Intelligence Tools, 20*(01), 209-219.

Harman, M., & Clark, J. (2004). Metrics are fitness functions too. *Proceedings of the International Software Metrics Symposium (METRICS 2004)*, 58-69. doi:10.1109/METRIC.2004.1357891

Harman, M., Clark, J., & Cinnéide, M. O. (2013). Dynamic adaptive search based software engineering needs fast approximate metrics. *Proceedings of 4th International Workshop on Emerging Trends in Software Metrics (WETSoM)*, 1-6. doi:10.1109/WETSoM.2013.6619329

Harman, M., Lakhotiaa, K., Singerb, J., Whiteb, D. R., & Yooa, S. (2012). Cloud engineering is search based optimization too. *Journal of Systems and Software, 86*(9), 2225-2241. doi:10.1016/j.jss.2012.10.027

Harman, M., McMinn, P., de Souza, J., & Yoo, S. (2012). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification* (Vol. 7007, pp. 1-59). Berlin, DE: Springer. doi:10.1007/978-3-642-25231-0_1

Harman, M., Swift, S., & Mahdavi, K. (2005). An empirical study of the robustness of two module clustering fitness functions. *Proceedings of the 7th annual Conference on Genetic and Evolutionary Computation (GECCO '2005), Washington D.C., USA, 25-29 June, 2005*, 1029-1036. doi:10.1145/1068009.1068184

Harman, M., & Tratt, L. (2007). Pareto optimal search based refactoring at the design level. *Proceedings of the 9th annual conference on Genetic and Evolutionary Computation (GECCO '2007), London, England, 07-11 July, 2007*, 1106-1113. doi:10.1145/1276958.1277176

He, Z., & Yen, G. (2014). Comparison of many-objective evolutionary algorithms using performance metrics ensemble. *Advances in Engineering Software, 76*(1), 1-8. doi:10.1016/j.advengsoft.2014.05.006

Hemati-Moghadam, I., & Ó Cinnéide, M. (2012). Automated refactoring using design differencing. *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR), Hungary, Szeged , 27-30 March 2012*, 43-52 doi:10.1109/CSMR.2012.15

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly, 28*(1), 75-105. doi:10.2753/MIS0742-1222240302

Hughes, E. J. (2005). Evolutionary many-objective optimisation: Many once or one many? *Proceedings of the 1st IEEE Congress on Evolutionary Computation (CEC), Edinburgh, Scotland, 2-5 September, 2005, 1*, 222-227. doi:10.1109/CEC.2005.1554688

Izurieta, C., & Bieman, J. M. (2007). How software designs decay: A pilot study of pattern evolution. *Proceedings of the first International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain, 20-21 September, 2007*, 449-451.

Izurieta, C., & Bieman, J. M. (2013). A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal, 21*(2), 289-323. doi:10.1007/s11219-012-9175-x

Jacobson, I., Christerson, M., Johnson, P., & Övergaard, G. (1992). *Object oriented software engeneering: A use case driven approach* (Vol. 1). Boston, MA, USA: Addison-Wesley Professional.

Jansen, A., Avgeriou, P., & van der Ven, J. S. (2009). Enriching software architecture documentation. *Journal of Systems and Software, 82*(8), 1232-1248. doi:10.1016/j.jss.2009.04.052

Johnson, D. B. (1975). Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing, 4*(1), 77-84. doi:10.1137/0204007

Jones, C. (1997). *Software quality: Analysis and guidelines for success* (2 ed., Vol. 1). Boston, MA, USA: Thomson Learning.

Kan, S. H. (2002). *Metrics and models in software quality engineering* (2 ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co.

Kasprzyk, J. R., Reed, P. M., Characklis, G. W., & Kirsch, B. R. (2012). Many-objective de Novo water supply portfolio planning under deep uncertainty. *Environmental Modelling & Software, 34*(1), 87-104. doi:10.1016/j.envsoft.2011.04.003

Khoshgoftaar, T. M., Khoshgoftaar, T. M., & Seliya, N. (2004). The necessity of assuring quality in software measurement data. *Proceedings of the 10th International Symposium on Software Metrics (METRICS 2004)*, 119-130. doi:10.1109/METRIC.2004.1357896

Köppen, M., & Yoshida, K. (2007). Many-objective particle swarm optimization by gradual leader selection. *Adaptive and Natural Computing Algorithms, 4431*, 323-331. doi:10.1007/978-3-540-71618-1_36

Koschke, R. (2008). Architecture reconstruction: Tutorial on reverse engineering to the architectural level. *Proceedings of the International Summer School on Software Engineering (ISSSE)*, 140–173. doi:10.1007/978-3-540-95888-8_6

Kukkonen, S., & Lampinen, J. (2005). GDE3: The third evolution step of generalized differential evolution. *Proceedings of the Congress on Evolutionary Computation (CEC). 1*, 443-450. doi:10.1109/CEC.2005.1554717

Lakos, J. (1996). *Large-scale C++ software design* (Vol. 1). Boston, MA: Addison-Wesley Professional.

Lehman, M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE, 68*(9), 1060-1076.

Lenth, R. V. (2001). Some practical guidelines for effective sample size determination. *The American Statistician, 55*(3), 187-193. doi:10.1198/000313001317098149

Li, H., & Zhang, Q. (2009). Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. *IEEE Transactions on Evolutionary Computation, 13*(2), 284-302. doi:10.1109/TEVC.2008.925798

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research, 21*(2), 498-516. doi:jstor.org/stable/169020

Luke, S. (2010). *Essentials of metaheuristics*. Retrieved from http://cs. gmu. edu/∼ sean/book/metaheuristics

Maffort, C., Valente, M. T., Anquetil, N., Hora, A., & Bigonha, M. (2013). Heuristics for discovering architectural violations. *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE'13)*, 222-231. doi:10.1109/WCRE.2013.6671297

Mahdavi, K., Harman, M., & Hierons, R. M. (2003). A multiple hill climbing approach to software module clustering. *Proceedings of the International Conference on Software Maintenance (ICSM 2003)*, 315-324. doi:10.1109/ICSM.2003.1235437

Malveau, R., & Mowbray, T. J. (2003). *Software Architect Bootcamp*. New York, NY: Prentice Hall Professional Technical Reference.

Mancoridis, S., Mitchell, B. S., Chen, Y., & Gansner, E. R. (1999). Bunch: A clustering tool for the recovery and maintenance of software system structures. *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'99)*, 50-59.

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.

*Proceedings of the 6th International Workshop on Program Comprehension (IWPC'98)*, 45-53.

Martin, R. (1994). OO design quality metrics - An analysis of dependencies. *Proceedings of the Workshop of Pragmatic and Theoretical Directions in Object-Oriented Software Metrics (OOPSLA)*, 1-8.

Martin, R. (2000). Design principles and design patterns. *Retrieved from http://www.objectmetor.com.*

Martin, R. (2008). *Clean code: A handbook of agile software craftsmanship*. New York, NY: Pearson Education.

Martin, R. (2011). *The clean coder: A code of conduct for professional programmers* (1 ed.). New York, NY: Prentice Hall.

McIlroy, M. D., Buxton, J., Naur, P., & Randell, B. (1968). Mass-produced software components. *Proceedings of the 1st International Conference on Software Engineering, Garmisch Patenkirchen, Germany*, 88-98.

Melton, H., & Tempero, E. (2007). An empirical study of cycles among classes in Java. *Empirical Software Engineering, 12*(4), 389-415.

Melville, N. P. (2010). Information systems innovation for environmental sustainability. *MIS Quarterly, 34*(1), 1-21.

Meyer, B. (1988). *Object-oriented software construction* (Vol. 2). New York, NY: Prentice Hall.

Mitchell. (2002). *A heuristic search approach to solving the software clustering problem*. Drexel University, Drexel, Philadelphia, (unpublished PhD thesis).

Mitchell, & Mancoridis. (2001a). Comparing the decompositions produced by software clustering algorithms using similarity measurements. *Proceedings of the International Conference on Software Maintenance (ICSM 2001)*, 744–753. doi:10.1109/ICSM.2001.972795

Mitchell, & Mancoridis. (2001b). Craft: A framework for evaluating software clustering results in the absence of benchmark decompositions. *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE 2001)*, 93-102.

Mitchell, & Mancoridis. (2006). On the automatic modularization of software systems using the Bunch tool. *IEEE Transactions on Software Engineering, 32*(3), 193-208. doi:10.1109/TSE.2006.31

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. doi:10.1007/s00500-007-0218-3

Mitchell, Traverso, & Mancoridis. (2001). An architecture for distributing the computation of software clustering algorithms. *Proceedings of the Working Conference on Software Architecture (WCSA)*, 181-190. doi:10.1109/WICSA.2001.948427

Mordal, K., Anquetil, N., Laval, J., Serebrenik, A., Vasilescu, B., & Ducasse, S. (2013). Software quality metrics aggregation in industry. *Journal of Software: Evolution and Process, 25*(10), 1117-1135. doi:10.1002/smr.1558

Mostaghim, S., & Schmeck, H. (2008). Distance based ranking in many-objective particle swarm optimization. *Proceedings of Parallel Problem Solving from Nature (PPSN)*, 753-762. doi:10.1007/978-3-540-87700-4_75

Murphy, G. C., & Notkin, D. (1997). Reengineering with reflexion models: A case study. *Computer, 30*(8), 29-36. doi:10.1109/2.607045

Murphy, G. C., Notkin, D., & Sullivan, K. J. (2002). Software reflexion models: Bridging the gap between design and implementation. *Software Engineering, IEEE Transactions on, 27*(4), 364-380.

Nakib, A., & Siarry, P. (2013). Performance analysis of dynamic optimization algorithms. In E. Alba, A. Nakib, & P. Siarry (Eds.), *Metaheuristics for dynamic optimization* (Vol. 433, pp. 1-16). Berlin, DE: Springer. doi:10.1007/978-3-642-30665-5_1

Nebro, A. J., Durillo, J., Garcia-Nieto, J., Coello Coello, C., Luna, F., & Alba, E. (2009). Smpso: A new pso-based metaheuristic for multi-objective optimization. *Proceedings of Conference on Computational intelligence in Multi-Criteria Decision-Making (MCDM '09)*, 66-73. doi:10.1109/MCDM.2009.4938830

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., & Alba, E. (2007). Design issues in a multiobjective cellular genetic algorithm. *Proceedings of 4th International Conference Evolutionary Multi-criterion Optimization (EMO)*, 126-140.

Nebro, A. J., Luna, F., Alba, E., Dorronsoro, B., Durillo, J. J., & Beham, A. (2008). AbYSS: Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation, 12*(4), 439-457. doi:10.1109/TEVC.2007.913109

O'Keeffe, M., & Cinnéide, M. Ó. (2006). Search-based software maintenance. *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR), 10*, 10-260. doi:10.1109/CSMR.2006.49

Okabe, T., Jin, Y., & Sendhoff, B. (2003). A critical survey of performance indices for multi-objective optimisation. *Proceedings of the Congress on Evolutionary Computation (CEC'03), 2*, 878-885. doi:10.1109/CEC.2003.1299759

Oyetoyan, T. D., Cruzes, D. S., & Conradi, R. (2013). A study of cyclic dependencies on defect profile of software components. *Journal of Systems and Software, 86*(12), 3162–3182. doi:10.1016/j.jss.2013.07.039

Pareto, V. (1896). Cours economie politique. *Journal of Political Economy, 6*(4), 549-552.

Parnas, D. L. (2011). Precise documentation: The key to better software. In S. Nanz (Ed.), *The Future of Software Engineering* (pp. 125-148). Berlin, DE: Springer. doi:10.1007/978-3-642-15187-3_8

Passos, L., Terra, R., Valente, M. T., Diniz, R., & Mendonça, N. (2010). Static architecture-conformance checking: An illustrative overview. *IEEE Software, 27*(5), 82-89. doi:10.1109/MS.2009.117

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems, 24*(3), 45-77. doi:10.2753/MIS0742-1222240302

Porras, G. C., & Guéhéneuc, Y.-G. (2010). An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering, 15*(5), 493-522. doi:10.1007/s10664-009-9125-9

Praditwong, Harman, & Yao. (2011). Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering, 37*(2), 264-282. doi:10.1109/TSE.2010.26

Praditwong, K., Harman, M., & Yao, X. (2011). Software module clustering as a multi-objective search problem. *IEEE Transactions on Software Engineering*.

Pruijt, L., & Brinkkemper, S. (2014). A metamodel for the support of semantically rich modular architectures in the context of static architecture compliance checking. *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA 2014)*, 1-8. doi:10.1145/2578128.2578233

Randell, B. (1996). *The 1968/69 NATO software engineering reports*. presented at the meeting of the Dagstuhl Seminar on: The History of Software Engineering, Schloss Dagstuhl, Leibniz, Germany 26 - 30 August 1996.

Razali, N. M., & Wah, Y. B. (2011). Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of Statistical Modeling and Analytics, 2*(1), 21-33.

Ribeiro, J. C. B., Zenha-Rela, M. A., & Fernández de Vega, F. (2009). Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software. *Information and Software Technology, 51*(11), 1534-1548. doi:10.1016/j.infsof.2009.06.009

Rodríguez, D., Ruiz, M., Riquelme, J. C., & Harrison, R. (2011). Multiobjective simulation optimisation in software project management. In *Proceedings of the 13th annual Conference on Genetic and Evolutionary Computation (GECCO)* (pp. 1883-1890): ACM. doi:10.1145/2001576.2001829

Sarro, F. (2011). Search-based approaches for software development effort estimation. *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement (Profes '11)*, 38-43. doi:10.1145/2181101.2181111

Sayyad, A. S., & Ammar, H. (2013). Pareto-optimal search-based software engineering (POSBSE): A literature survey. *Proceedings of the 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, 21-27. doi:10.1109/RAISE.2013.6615200

Sayyad, A. S., Menzies, T., & Ammar, H. (2013). On the value of user preferences in search-based software engineering: A case study in software product lines. *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, 492-501.

Schmidt, F., MacDonell, S. G., & Connor, A. M. (2012). An automatic architecture reconstruction and refactoring framework. In R. Lee (Ed.), *Software Engineering Research, Management and Applications 2012* (Vol. 377, pp. 95-111). Berlin, DE: Springer. doi:10.1007/978-3-642-23202-2_7

Seng, O., Bauer, M., Biehl, M., & Pache, G. (2005). Search-based improvement of subsystem decompositions. *Proceedings of the 7th annual Conference on Genetic and Evolutionary Computation (GECCO 2005)*, 1045-1051. doi:10.1145/1068009.1068186

Sheskin, D. J. (2003). *Handbook of parametric and nonparametric statistical procedures* (Vol. 1). London, UK: Chapman and Hall.

Sierra, M. R., & Coello, C. A. C. (2005). Improving PSO-based multi-objective optimization using crowding, mutation and ∈-dominance. *Proceedings of the third international conference on Evolutionary Multi-Criterion Optimization (EMO)*, 505-519. doi:10.1007/978-3-540-31880-4_35

Sora, I., Glodean, G., & Gligor, M. (2010). Software architecture reconstruction: An approach based on combining graph clustering and partitioning. *Proceedings of the International Joint Conference on Computational Cybernetics and Technical Informatics (ICCC-CONTI)*, 259-264. doi:10.1109/ICCCYB.2010.5491289

Szwarcfiter, J. L., & Lauer, P. E. (1976). A search strategy for the elementary cycles of a directed graph. *BIT Numerical Mathematics, 16*(2), 192-204. doi:10.1007/BF01931370

Szyperski, C., Bosch, J., & Weck, W. (1999). Component-oriented programming. *Proceedings of the Conference on Object-Oriented Technology (ECOOP'99)*, 184-192. doi:10.1007/3-540-46589-8_10

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation* (Vol. 1). Hoboken, N.J.: John Wiley & Sons.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing, 1*(2), 146-160. doi:10.1137/0201010

Tarjan, R. (1973). Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing, 2*(3), 211-216. doi:10.1137/0202017

Taylor, R. N., Medvidovic, N., & Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice* (Vol. 1). New York, NY: Wiley.

Terra, R., & Valente, M. T. (2009). A dependency constraint language to manage object-oriented software architectures. *Software: Practice and Experience, 39*(12), 1073-1094.

Terra, R., Valente, M. T., Bigonha, R. S., & Czarnecki, K. (2012). DCLfix: A recommendation system for repairing architectural violations. *Proceedings of the Conference on Software: Theory and Practice (CBSoft)*, 1-6.

Tiernan, J. C. (1970). An efficient search algorithm to find the elementary circuits of a graph. *Communications of the ACM, 13*(12), 722-726. doi:10.1145/362814.362819

Tucker, A., Swift, S., & Liu, X. (2001). Variable grouping in multivariate time series via correlation. *IEEE Transactions on Systems and Cybernetics, 31*(2), 235-245. doi:10.1109/3477.915346

Vaishnavi, V., & Kuechler, W. (2008). *Design science research methods and patterns: Innovating information and communication technology* (Vol. 1). Boca Raton, FL: Auerbach Publications. doi:10.1016/S0164-1212(01)00152-2

Van Gurp, J., & Bosch, J. (2002). Design erosion: Problems and causes. *Journal of Systems and Software, 61*(2), 105-119. doi:10.1016/S0164-1212(01)00152-2

Van Heesch, U., Avgeriou, P., & Hilliard, R. (2012). A documentation framework for architecture decisions. *Journal of Systems and Software, 85*(4), 795-820. doi:10.1016/j.jss.2011.10.017

Van Veldhuizen, D. A., & Lamont, G. B. (1999). Multiobjective evolutionary algorithm test suites. *Proceedings of the ACM Symposium on Applied Computing (SAC '99)*, 351-357. doi:10.1145/298151.298382

Van Veldhuizen, D. A., & Lamont, G. B. (2000). On measuring multiobjective evolutionary algorithm performance. *Proceedings of the first Congress on Evolutionary Computation (CEC). 1*, 204-211. doi:10.1109/CEC.2000.870296

Wegener, J., Baresel, A., & Sthamer, H. (2001). Evolutionary test environment for automatic structural testing. *Information and Software Technology, 43*(14), 841-854. doi:10.1016/S0950-5849(01)00190-2

Yang, X. S. (2008). *Introduction to mathematical optimization* (Vol. 1). Cambridge, UK: Cambridge International Science Publishing.

Yu, L., & Mishra, A. (2013). An Empirical Study of Lehman's Law on Software Quality Evolution. *Int J Software Informatics, 7*(3), 469-481.

Zitzler, E., Brockhoff, D., & Thiele, L. (2007). The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. *Proceedings of the 4th International Evolutionary Multi-Criterion Optimization Conference (EMO)*, 862-876. doi:10.1007/978-3-540-70928-2_64

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the strength Pareto evolutionary algorithm*. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK), Zuerich, CH.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation, 7*(2), 117-132. doi:10.1109/TEVC.2003.810758

# Appendix A: Computational Resources

The experiments (search and evaluation analysis) are executed on two virtual machines that have been hosted on a Virtual Machine (VM) farm. The two VM's feature the following configuration:

- 24 (6x4) virtual cores (Opteron – 6348)
- 256 GB memory
- Ubuntu 12.04 Operating System

# Appendix B: Architecture Configuration Instance

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE project SYSTEM "Configuration.dtd">
<project
  name="apache log4j"

      <!-- DIRECTORIES -->
            <input dir="../target/classes/"/>

      <!-- SKIP -->
      <skip prefix="java.lang.[a-zA-Z]*"/>
      <skip prefix="java.io.*"/>
      <skip prefix="java.text.*"/>
      <skip prefix="java.util.*"/>

      <!-- LOGICAL ARCHITECTURE -->
      <layer name="bottom">
            <description>bottom Layer</description>
            <subsystem name="application">
                  <description></description>
            </subsystem>
      </layer>
      <layer name="mid">
            <description>Service Layer</description>
            <subsystem name="application">
                  <description></description>
                  <depends-upon name="bottom::application"/>
            </subsystem>
      </layer>
      <layer name="mid2">
            <description>Service Layer</description>
            <subsystem name="application">
                  <description></description>
                  <depends-upon name="bottom::application"/>
            <depends-upon name="mid::application"/>
            </subsystem>
      </layer>
      <layer name="top">
            <description>top layer</description>
            <subsystem name="application">
                  <description></description>
            <depends-upon name="bottom::application"/>
            <depends-upon name="mid::application"/>
              <depends-upon name="mid2::application"/>
            </subsystem>
      </layer>
</project>
```

# Appendix C: Instance of *ExperimentConfiguration*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<experimentConfiguration>
    <algorithmConfigurationContainer>
        <algorithmName>NSGAII</algorithmName>
        <fromNumberOfIterations>50000</fromNumberOfIterations>
        <fromPopulationSize>200</fromPopulationSize>
        <numberOfSeeds>12</numberOfSeeds>
        <numberOfThreads>12</numberOfThreads>
        <variationsOperators>
            <_variationPropertyTupel>
                <variationPropertyTupel>
                    <fromValue>0.5</fromValue>
                    <name>pm.rate</name>
                    <toValue>0.0</toValue>
                </variationPropertyTupel>
                <variationPropertyTupel>
                    <fromValue>10.0</fromValue>
                    <name>pm.distributionIndex</name>
                    <toValue>0.0</toValue>
                </variationPropertyTupel>
            </_variationPropertyTupel>
            <_isCrossover>false</_isCrossover>
            <_name>Polynomial Mutation</_name>
        </variationsOperators>
        <variationsOperators>
            <_variationPropertyTupel>
                <variationPropertyTupel>
                    <fromValue>1.0</fromValue>
                    <name>sbx.rate</name>
                    <toValue>0.0</toValue>
                </variationPropertyTupel>
                <variationPropertyTupel>
                    <fromValue>10.0</fromValue>
                    <name>sbx.distributionIndex</name>
                    <toValue>0.0</toValue>
                </variationPropertyTupel>
            </_variationPropertyTupel>
            <_isCrossover>true</_isCrossover>
            <_name>Simulated Binary Crossover</_name>
        </variationsOperators>
     </algorithmConfigurationContainer>
     <optimisationGoals>
        <numberInResultFile>0</numberInResultFile>
        <objective>Minimize</objective>
        <rearchitecturerMetric>
            <_metric>

           <_key>NumberOfCyclesInPackages_project_apache_log4j</_key>
                <_name>Number of Cycles in Packages</_name>
                <_entitiyType>Project</_entitiyType>
                <_entitiyName>apache_log4j</_entitiyName>
                <_simpleName>NumberOfCyclesInPackages</_simpleName>
                <name>NumberOfCyclesInPackages</name>
            </_metric>
```

```
        </rearchitecturerMetric>
    </optimisationGoals>
    <optimisationGoals>
        <numberInResultFile>0</numberInResultFile>
        <objective>Minimize</objective>
        <rearchitecturerMetric>
            <_metric>
                <_key>NCCD_project_apache_log4j</_key>
                <_name>NCCD</_name>
                <_entitiyType>Project</_entitiyType>
                <_entitiyName>apache_log4j</_entitiyName>
                <_simpleName>NCCD</_simpleName>
                <name>NCCD</name>
            </_metric>
        </rearchitecturerMetric>
    </optimisationGoals>
    <project>Apache Log4j</project>
    <reconstructionConfiguration>
        <reassignCompilationsRandomlyAtSeedStart>false
        </reassignCompilationsRandomlyAtSeedStart>
        <_assignCompilationsUnits>true</_assignCompilationsUnits>
        <_assignPackages>true</_assignPackages>
        <_maxNumberofPackages>3</_maxNumberofPackages>
        <_maxNumberofSubsystems>3</_maxNumberofSubsystems>
        <_redevelopPackageStructure>false
        </_redevelopPackageStructure>
        <_redevelopSubsystemStructure>false
        </_redevelopSubsystemStructure>
    </reconstructionConfiguration>
    <reconstructionName>AssignCompilationUnitsAndPackages
    </reconstructionName>
    <recordAccuraccyMetrics>false</recordAccuraccyMetrics>
    <recordDecisionVariableValues>false
    </recordDecisionVariableValues>
    <trackArchitectureSolutions>false</trackArchitectureSolutions>
    <_enableLogging>false</_enableLogging>
    <_keepPreviousDefinedPackageToSubsystemAssignments>false
    </_keepPreviousDefinedPackageToSubsystemAssignments>
    <_numberOfSubsystemDependencies>3
    </_numberOfSubsystemDependencies>
</experimentConfiguration>
```

# Appendix D: Instance of *MetaExperimentConfiguration*

```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <metaExperimentConfiguration>
       <_algorithms>OMOPSO</_algorithms>
       <_algorithms>AbYSS</_algorithms>
       <_algorithms>NSGAII</_algorithms>
       <_algorithms>MOEAD</_algorithms>
       <_algorithms>GDE3</_algorithms>
       <_algorithms>Random</_algorithms>
           <_configurations>
           <_architectureConfiguration>/path/to/apache-ant-
1.9.2/architectureConfiguration.xml
           </_architectureConfiguration>
           <_metaheuriticConfiguration>/path/to/apache-ant-
1.9.2/experimentConfiguration.xml
           </_metaheuriticConfiguration>
           </_configurations>
              <_configurations>
           <_architectureConfiguration>/path/to/apache-log4j-
1.2.17/architectureConfiguration.xml
           </_architectureConfiguration>
           <_metaheuriticConfiguration>/path/to/apache-log4j-
1.2.17/experimentConfiguration.xml
           </_metaheuriticConfiguration>
           </_configurations>
       <_numberOfIterations>55000</_numberOfIterations>
       <_numberOfParallelExecutions>20</_numberOfParallelExecutions>
       <_numberofPackageAssignments>0</_numberofPackageAssignments>
       <_numberofSeeds>4</_numberofSeeds>
       <_numberofVariations>3</_numberofVariations>

<_subDir>strict_keep_assignments_pm_rate_05_sbx_rate_10_pm_di_10_sbx_di
_10_8_objectives</_subDir>
   </metaExperimentConfiguration>
```

# Appendix E: Batch Driven Execution of Search Configurations

The main functionality to execute experiment scenarios in batch mode is implemented in the class batch.RearchitecturerExperimentExecuter. A reference to a JAXB-xml file instance of the type MetaExperimentConfiguration is required as a parameter to execute the creation of data sets. The MetaExperimentConfiguration instance defines:

- A reference to an ExperimentConfiguration instance
- A set of systems (references to architecture descriptions files)
- A set of MOEA implementations
- The number of seeds (reruns)
- A target result directory
- The number of cores that are utilised in the search

The ExperimentConfiguration instance describes the configuration of the search such as objective configuration, variation operator settings and reconstruction methods. Appendix C depicts an example of an ExperimentConfiguration instance. The ExperimentConfiguration instance can be executed independently, for example, by loading it into the GUI component. However, this study is mainly interested in comparative evaluations of different configuration settings. Hence, the ExperimentConfiguration instance is executed with each system configuration and MOEA implementation that is defined in the MetaExperimentConfiguration instance. Appendix D depicts an example of a MetaExperimentConfiguration instance. Each of these constellations is a separate search configuration. The number of seeds defines the number of reruns of these search configurations.

The employment of a search configuration in the Rearchitecturer component results in a solution set. A solution set contains a header line with the names of the objectives and the corresponding objective values of the visited solutions of an experiment configuration run. Hence, a solution set represents a seed of the execution of a search configuration. The objective names and objective values are separated by commas (csv format).

The Rearchitecturer artefact stores the solution sets (seeds) with the corresponding ExperimentConfiguration and ArchitectureConfiguration instance in a subdirectory of the target result directory. Hence, all solution sets in one of these subdirectories have been created based on the same search configuration. The batch.RearchitecturerExperimentExecuter implementation enables the execution of configuration runs in parallel based on the number of cores that are defined in the MetaExperimentConfiguration instance. The MetaExperimentConfiguration instances that have been employed in the data collection and the scripts to execute the individual evaluation scenarios of this research can be downloaded from the project webpage.

# Appendix F: Employment of Search Configuration Analysis

The class analysis.RearchitecturerExperimentExecuter implements the main method to conduct statistical analysis of solution sets. A reference to a JAXB-xml file instance of the type AnalysisConfiguration is required as a parameter to execute the analysis of data sets. Appendix:G depicts an example of an AnalysisConfiguration instance. The AnalysisConfiguration instance defines:

- A set of input directories of solution sets
- A target directory of the result files of the analysis
- Number of cores that are utilised in the analysis
- Frequency of analysis snapshots
- Termination criteria (solution number)
- Method of Analysis
- NDPF classification criteria
- Performance snapshot range

The AnalysisConfiguration instance allows the user to control different aspects of the analysis. The following sections elaborate on the functionality of the analysis framework and relate the functionality aspects to the corresponding configuration parameters. The set of AnalysisConfiguration instances that are employed in the evaluation of this research are available on the Rearchitecturer project website.

# Appendix G: Instance of *AnalysisConfiguration*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<analysisConfiguration>
    <algorithms>Random</algorithms>
    <algorithms>NSGAII</algorithms>
    <algorithms>GDE3</algorithms>
    <algorithms>OMOPSO</algorithms>
    <algorithms>MOEAD</algorithms>
    <algorithms>AbYSS</algorithms>
    <systems>
        <entry>
         <key>Apache Ant</key>
         <value>/path/to/output/for/project/apache_log4j/</value>
         </entry>
      <entry>
         <key>Lucene</key>
         <value>/path/to/project/results/lucene/</value>
         </entry>
    </systems>
    <_frequency>200</_frequency>
    <_metaResultsOutputDir>/>/path/to/results/metaresults
    </_metaResultsOutputDir>
    <_numberOfIterations>50000</_numberOfIterations>
    <_resultInputDir>/path/to/project/solutionsets/apache_log4j
    </_resultInputDir>
    <_resultInputDir>/path/to/project/solutionsets/lucene
    </_resultInputDir>
    <_numberOfCores>16</_numberOfCores>
    <paretoFrontMetrics>true</paretoFrontMetrics>
    <performanceMetrics>true</performanceMetrics>
    <objectiveMetrics>true</objectiveMetrics>
</analysisConfiguration>
```