Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Techniques for Visualization and Interaction in Software Architecture Optimization

Sebastian Frank

Course of Study:	Softwaretechnik
Examiner:	Dr.-Ing. André van Hoorn
Supervisor:	Dr.-Ing. André van Hoorn Dr. J. Andrés Díaz-Pace, UNICEN, ARG Dr. Santiago Vidal, UNICEN, ARG
Commenced:	March 25, 2019
Completed:	September 25, 2019
CR-Classification:	D.2.11, H.5.2, H.1.2

Abstract

Software architecture optimization aims at improving the architecture of software systems with regard to a set of quality attributes, e.g., performance, reliability, and modifiability. However, particular tasks in the optimization process are hard to automate. For this reason, architects have to participate in the optimization process, e.g., by making trade-offs and selecting acceptable architectural proposals.

The existing software architecture optimization approaches only offer limited support in assisting architects in the necessary tasks by visualizing the architectural proposals. In the best case, these approaches provide very basic visualizations, but often results are only delivered in textual form, which does not allow for an efficient assessment by humans.

Hence, this work investigates strategies and techniques to assist architects in specific use cases of software architecture optimization through visualization and interaction. Based on this, an approach to assist architects in these use cases is proposed. A prototype of the proposed approach has been implemented.

Domain experts solved tasks based on two case studies. The results show that the approach can assist architects in some of the typical use cases of the domain. Conducted time measurements indicate that several hundred architectural proposals can be handled. Therefore, the approach usefully complements existing software architecture optimization approaches. Furthermore, it is a foundation for the participation of humans in the optimization process and further research in this field.

Kurzfassung

Softwarearchitekturoptimierung hat die Verbesserung der Architekturen von Softwaresystemen bezüglich einer Auswahl von Qualitätsattributen, z. B. Leistungsfähigkeit, Zuverlässigkeit und Änderbarkeit, zum Ziel. Allerdings lassen sich einige Aufgaben im Optimierungsprozess nur schwer automatisieren. Aus diesem Grund müssen Softwarearchitekten in den Optimierungsprozess eingebunden werden, z. B. indem sie Kompromisse eingehen und akzeptierbare Architekturvorschläge auswählen.

Die existierenden Ansätze der Softwarearchitekturoptimierung bieten den Softwarearchitekten nur begrenzte Hilfe durch Visualisierung der Architekturvorschläge, um die notwendigen Aufgaben zu erfüllen. Im besten Fall stehen sehr schlichte Visualisierungen zur Verfügung, aber häufig werden die Resultate nur in textueller Form bereitgestellt, welche nicht für eine effiziente Begutachtung durch Menschen geeignet ist.

Deshalb untersucht diese Arbeit Strategien und Techniken, um Softwarearchitekten in den für Softwarearchitekturoptimierung relevanten Anwendungsfällen durch Visualisierungen und Interaktion zu unterstützen. Basierend darauf wird ein Ansatz zur Unterstützung von Softwarearchitekten in diesen Anwendungsfällen vorgestellt. Ein Prototyp des vorgeschlagenen Ansatzes wurde hierzu implementiert.

Fachexperten mussten Aufgaben basierend auf zwei Fallstudien lösen. Die Ergebnisse zeigen, dass der vorgeschlagene Ansatz geeignet ist, um Softwarearchitekten in einigen domänenspezifischen Anwendungsfällen zu unterstützen. Durchgeführte Zeitmessungen deuten darauf hin, dass mehrere hundert Architekturvorschläge verarbeitet werden können. Aus diesem Grund ist der präsentierte Ansatz eine nützliche Ergänzung zu den existierenden Softwarearchitekturoptimierungsmethoden. Darüber hinaus ist er eine Grundlage für die Einbindung von Menschen in den Optimierungsprozess und weitere Forschung diesbezüglich.

Contents

List of Figures

List of Tables

List of Acronyms

AADL Architecture Analysis and Design Language

Ajax asynchronous JavaScript and XML

API application programming interface

AQOSA Automated Quality-driven Optimization of Software Architecture

ArchE Architecture Expert

CPU central processing unit

CSS Cascading Style Sheets

CSV comma-separated values

DAO data access object

HDD hard disk drive

HTML Hypertext Markup Language

JPA Java Persistence API

JSF JavaServer Faces

PCM Palladio Component Model

ROBOCOP Robust Open Component Based Software Architecture for Configurable Devices Project

SQuAT Search Techniques for Managing Quality-Attribute Tradeoffs in Software Design Optimizations

TCP Transmission Control Protocol

UC use case

UML Unified Modeling Language

Chapter 1

Introduction

The introduction is divided into four sections. Section 1.1 explains the need for software architecture optimization. Then, Section 1.2 describes some of the challenges in the domain of software architecture optimization. Section 1.3 outlines the benefits of a visualization approach for the domain of software architecture optimization. Additionally, it summarizes how this work solves this task. Finally, the structure of this work is described in Section 1.4.

1.1. Motivation for Software Architecture Optimization

Quality attributes, like performance and modifiability, play an important role in software engineering [Gil92]. Thus, the stakeholders of a software system state not only functional but also non-functional requirements for the system. For example, the users are often interested in a good performance of the system [SW03]. Performance is especially important for web applications [Nah04]. Case studies and experiments by companies like Google emphasize this importance. For example, one case study shows that an increase in Google's web search latency by 100 to 400 ms can lead to a reduction of the daily number of searches by 0.2% to 0.6% [Bru09]. Thus, companies have a big interest in the performance of their software system, but also in other quality attributes. Maintenance does often contribute 40% to 80% of the costs to a software system [Gla01]. Therefore, the modifiability of the system can have a big impact on the financial profit of the companies [Mun78].

Satisfying these non-functional requirements can be a challenging task for software architects due to conflicts [BI96]. Even a single quality attribute can be influenced by several design decisions, e.g., performance can be influenced by the allocation of software components to servers and the chosen hardware. For this reason, the

number of possible configurations quickly becomes too big to be efficiently explored by a human. In addition, changes to the architecture can improve one quality attribute, while another quality attribute deteriorates at the same time. For example, a cache can be implemented to improve the system's performance, but the additional code decreases the maintainability of the system.

It becomes obvious that a human architect is usually unable to find the best architecture for a big software system with many requirements on conflicting quality attributes. For this reason, software architecture optimization approaches offer automated or semi-automated assistance. However, an increasing number of quality attributes, non-functional requirements, and degrees of freedom in the modeled software system make a complete exploration of the search space infeasible, even with such tools [Ale+13]. A common strategy to handle this problem is the use of heuristics and the reduction of the number of solutions presented to the architect. While a high degree of automation in this process is more convenient for the architect, it also becomes more challenging for the tool to find a satisfying solution. Conflicts between quality attributes or requirements can usually not be resolved by the tool alone. Thus, the architect has to express her preferences at least at one point in the optimization process. There are three possible times at which the architect can state her preferences [Bra+08; VVL00]. One possibility is to ask the architect for her preferences before the start of the optimization, but this comes with the risk of wasting a lot of time and effort during the search. This is due to the fact that the architect's preferences can be changed in the process of the optimization. Then, the proposed solution can not fit the architect's expectations. Another possibility is to ask the user after the optimization to evaluate different potential solutions, but this can also mean that the tool wasted a lot of time searching for solutions, which the user was never interested in. Finally, it is also possible to ask the architect multiple times to evaluate the system's proposal candidates. The architect can then react to new insights during the optimization process and guide the tool in the desired direction.

1.2. Challenges in Software Architecture Optimization

A software architecture approach that emphasizes this kind of user interaction is SQuAT [Rag+17] (Search Techniques for Managing Quality-Attribute Tradeoffs in Software Design Optimizations), which (currently) uses the Palladio Component Model (PCM) [Reu+11] as a representation of software architectures. Independent modules, so-called bots, are able to evaluate and optimize candidates with regard to exactly one quality attribute and goal, e.g., the architect could state the goal that the system should still respond after at most one second if the amount of users doubles. After each bot tried to optimize the initial candidate, the bots negotiate in order to find a candidate

that (mostly) satisfies the requirements of the architect. The approach requires some configuration by the architect, e.g., specifying goals, which are called scenarios [CKK02], and expected response values. Additionally, it is possible to continue the search and use selected candidates as input for a new iteration of the search. In theory, this is a suitable point for interaction between the architect and SQuAT. The architect could select promising candidates for the next iteration or change the configuration based on new insights.

However, it is currently a hard and time-consuming task for the architect to assess the proposed candidates because the output of SQuAT is currently completely textual. This is a problem that is not specific to SQuAT, but common for all the major software architecture optimization approaches, which all provide at most a limited support for the (visual) assessment of the proposed candidates. These are usually not sufficient to help the architect in making decisions within a reasonable time. Techniques from the domain of information visualization might solve this issue as "information visualization utilizes computer graphics and interaction to assist humans in solving problems" [Pur+08], such as selecting promising architectural candidates.

The architect could also use scripts and/or available visualization tools, e.g., the Trade Space Visualizer [Stu+04], to solve this problem. However, creating scripts takes time and architects might not be willing to do that or do not have the skills. Generic visualization approach can help, but they are not designed for the use in software architecture optimization and are unlikely to be able to answer all the questions that architects might have. Additionally, they also do not support direct interaction with software architecture optimization approaches.

In order to choose architectures as a solution or input for the next level of search, the architect needs to compare the architectures. Comparisons of all or many candidates at the same time can give an overview and help to make a rough selection. A detailed comparison between the candidates of this selection can be used to further reduce the selection. These comparisons can not only be made based on the quality values but also on the architecture. Information about the architecture can help the architect to understand which architectural changes have a significant impact on the quality values, e.g., she could see that almost all candidates with good performance contain a cache. This knowledge can help the architect to adopt this change in the real system and make informed decisions in the optimization process. In addition, it also makes the optimization process itself more comprehensible to the architect. This effect should not be underestimated, as it increases the trust of the architect in the software architecture optimization tool. And trust, according to Lee et al. [LS04], is important when it comes to the automation of tasks. The architect is more likely to accept the proposed architecture if she can comprehend what the tool actually did. This also requires information about how candidates evolved.

1.3. Visualization Approach for Software Architecture Optimization

For the previously stated reasons, an approach that supports visualization and interaction specific to software architecture optimization is a useful contribution to this field. It does not only interconnect the domains of information visualization and software architecture optimization, but could also be used to supplement software architecture optimization approaches. This would make it easier for architects to explore the results. It is also a foundation for the participation of humans in the optimization process. Additionally, architects would be able to react to new insights and handling huge design spaces becomes easier.

Therefore, this work presents such an approach that supports the architect in typical use cases of software architecture optimization. This includes selecting candidates and explaining results, but also finding a stopping criterion for the optimization and identifying necessary changes for the implementation of proposed solutions. Suitable techniques for visualization and interaction are presented to achieve this goal. In addition, the presented approach supports interaction, which allows the user to efficiently explore the results and participate in the optimization process by selecting promising architectures for further optimization. A prototype has been developed for SQuAT [Rag+17] and PCM [Reu+11]. However, it is also modular and generic. Thus, it could be used in similar approaches with limited efforts. The results gathered in an expert user study indicate that the approach reaches most of the previously described goals.

1.4. Thesis Structure

This work is structured in the following way:

Chapter 2 – Foundations: This chapter describes the foundations of this work. The main topics are software architecture optimization, information visualization, and interactive optimization. It also outlines the SQuAT approach and the available prototype.

Chapter 3 – Related Work: An overview of works that are related to this work. The similarities and differences are outlined.

Chapter 4 – Concept: The requirements and solutions for an approach to visualize the results of software architecture optimization are described here. This includes a description of the used visualizations and interaction techniques that are assembled in different views.

Chapter 5 – Prototype: This chapter focuses on the prototype. It outlines the technical requirements, used technologies, and design of the prototypical implementation.

Chapter 6 – Evaluation: Describes the experimental setup and the results of the conducted expert review and the time measurements.

Chapter 7 – Conclusion: Summarizes this work and outlines potential future work.

The measurements and results of the expert review that have been conducted in the evaluation are publicly available as part of the work's supplementary material [Frab]. The original code base of this work is also publicly available [Fraa; Frab].

Chapter 2

Foundations

This chapter gives an overview of basic concepts and techniques that are relevant to this work. Therefore, it outlines the state of the art in the relevant domains by highlighting important works. Section 2.1 focuses on the foundations in software architecture optimization itself. Since this work should design or select suitable techniques and strategies for visualization and interaction, a selection of concepts in the domain of information visualization is examined in more detail in Section 2.2.

2.1. Software Architecture Optimization

The overall purpose of software architecture optimization is to select an architecture for a software system that meets the architect's quality requirements. Therefore, optimization usually has one or more defined goals. Section 2.1.1 outlines these goals in more detail. In order to optimize software architectures, software models are usually used to represent the software system. Thus, software modeling is examined in more detail in Section 2.1.2. The optimization process itself is presented in Section 2.1.3. Finally, Section 2.1.4 lists some state-of-the-art approaches for multi-objective software architecture optimization.

2.1.1. Optimization Goals

Software systems are usually designed for a specific purpose, which is described by functional requirements. These functional requirements certainly influence the design and the architecture of the software system. However, there are also many degrees of freedom in software architectures that do usually not influence the functionality, e.g., the allocation of components to servers. However, these degrees of freedom can

still influence the quality of a software system, e.g., the allocation of components can influence how quickly a response is delivered to the user of a software system. Therefore, non-functional software quality is also an important property of a software system for the stakeholders [Gil92].

For this reason, software architects often need to choose a software architecture that does not only meet functional but also non-functional requirements. These non-functional requirements, in general, describe the important quality attributes of the system, e.g., performance or modifiability. Therefore, architects in software architecture optimization define their goals based on these quality attributes. According to Aleti et al. [Ale+13], the software quality attributes performance, cost, and reliability are investigated most often. However, there are also quality attributes that are only relevant to some architects, e.g., energy consumption. The choice and the number of relevant quality attributes depends on the particular software system, its environment, and the preferences of its stakeholders.

To evaluate the quality of a software system, metrics have to be defined. For example, common metrics for performance are response time, throughput, and resource utilization [Jai90]. It has to be noted that these metrics sometimes have to be optimized in different directions, e.g., response time should be minimized, while throughput should be maximized. Yau and Chang [YC88] suggest to use the number of dependencies in a software system as a metric for its modifiability. The cyclomatic complexity [McC76] is another metric that quantifies modifiability. As different metrics for quality attributes exist, the choice of the metric is an essential decision for software architecture optimization.

The exact formulation of the goal is also important. The goal can simply be to reach the highest or lowest possible value. In this case, the degree of satisfaction increases when the value is improved. It is also possible to define a goal as constraint, then there is an expected value that has to be reached. In this case, there are just two options: the goal is fulfilled or not fulfilled. Goals can also be defined in the form of so-called scenarios [CKK02]. A scenario always describes a stimulus to an artifact in a particular environment induced by a source, e.g., an increase in the number of users puts a software system under higher load. The artifact then responds to this stimulus, which can be quantified by a response measure. In the previous example, the increase could lead to a degradation of the system's performance, which could be measured by the system's response time. Hence, a goal can be formulated as a scenario by stating a condition and an expected response.

In practice, often more than one goal is defined. Quality attributes, however, can conflict with other quality attributes, e.g., implementing a cache can improve performance, while the additional code decreases modifiability. This conflict can only be solved by the architect, who has to state her preferences. This has to be done either before,

during, or after the optimization [Koz14]. At the end of the optimization, a Pareto analysis [NZES05] can be conducted to identify solutions that are not dominated by others to limit the number of proposed solutions.

2.1.2. Software Modeling

Modifying and evaluating real software systems is, in general, infeasible due to the huge design space in software architecture optimization. Besides, many software architecture optimization approaches are designed for decision-making at design time [Ale+13] when the system is not implemented yet. Therefore, software architecture optimization approaches rely on models of the software systems to apply transformations and make predictions about the quality of the systems.

In general, all kinds of models can be used for software architecture optimization. This includes models that do not directly describe the architecture, e.g., Markov chain [KS76]. Approaches might even internally transform the input model to be able to evaluate its quality with regard to particular quality attributes and metrics. However, sophisticated software architecture optimization approaches usually use a more general architectural description in order to support the prediction of different quality attributes. Some software architecture optimization approaches [Gie+03] use the widespread Unified Modeling Language (UML) [RBJ17] for this purpose. It can provide structural and behavioral viewpoints on software systems and their environment. However, UML can be ambiguous and does not explicitly focus on the modeling of software systems. Examples of specific modeling languages are the Architecture Analysis and Design Language (AADL) [FG12] and the Palladio Component Model (PCM) [Reu+11]. Another one has also been developed in the Robust Open Component Based Software Architecture for Configurable Devices Project (ROBOCOP) [BC+06].

The Palladio Component Model (PCM) [Reu+11] is not only suited for the representation of component-based software systems, it has also been designed to predict the quality of the modeled system. The model itself consists of five parts that can be specified by different domain experts. In the *repository model*, components, interfaces, and the internal behavior of components are specified. The *system model* describes the dependencies between these components and the system's static structure. In the *resource model*, the available hardware can be specified, e.g., servers, central processing units (CPUs), and hard disk drive (HDDs). The *allocation model* describes to which servers the software components are allocated to. Finally, a *usage model* can be created to describe how the system is utilized by its users.

Due to the different viewpoints and features of PCM, it is suitable for the prediction of performance, reliability, cost, and maintainability, among others. For the quality predic-

tion, approaches that apply model-to-code transformations exist. Then, they simulate the system, e.g., the discrete-event simulator EventSim [MH11]. In addition, model-to-model transformations, e.g., a transformation to layered queueing networks [KR08], allows the use of solvers.

2.1.3. Optimization Process

Aleti et al. [Ale+13] describe the optimization process in software architecture optimization as illustrated in Figure 4.14. The input to the optimization process is always an architecture representation, which describes the initial architecture of the system. This can be the model of an existing software system or the design of a planned software system. At the end of the optimization process, the optimized architecture designs are returned. This can be a single one if single-objective optimization is applied or the preferences of the architect are known to the approach. Multiple architectures are usually returned in multi-objective optimization if the architect is expected to make final decision.

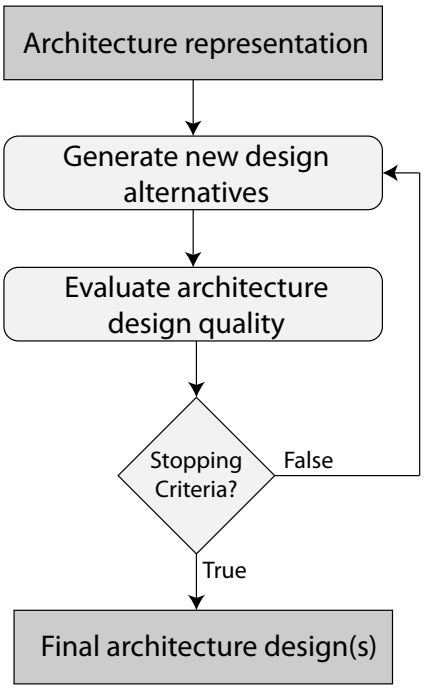

Figure 2.1.: Optimization process in software architecture optimization. [Ale+13]

The internal process consists of three steps. In the first step, new design alternatives have to be generated. Therefore, the approach must be equipped with transformations which make changes to the architecture. In general, transformations can be applied as

part of domain-specific rules or generic operations. For example, approaches based on genetic algorithms [CLV06] can apply mutation and crossover operations to create new design alternatives. These operations do not require domain specific knowledge.

Then, the approaches have to evaluate the generated alternatives with regard to the examined quality attributes. As explained in Section 2.1.2, simulators and solvers can be used to predict the quality of the systems through specified quality metrics. Finally, it has to be decided whether to continue the optimization or to stop it. If an alternative meets the stated requirements, the optimization is usually stopped. If it does not, new alternatives can be generated. In this process, also alternatives from previous iterations can be used as input. However, there can also be other stopping criteria, e.g., usually a maximum number of iterations is defined to assure termination. More sophisticated stopping criteria exist, e.g., PerOpteryx [KR11; Koz14] can stop the optimization if no or only little improvements are made within a specified number of iterations.

2.1.4. State-of-the-art Approaches

Many approaches for software architecture optimization exist. According to Aleti et al. [Ale+13], most of these approaches rely on heuristics, because the design space is too large for a complete search. They can be categorized based on the supported quality attributes, the underlying modeling language, and their optimization strategies. Some representative examples for the domain of multi-objective software architecture optimization are outlined in the following.

AQOSA [Li+11] consists of tools for software architecture optimization in the domain of embedded and component-based systems. These systems can be analyzed with regard to performance, reliability, and cost. Multiple evolutionary optimization algorithms have been implemented for AQOSA.

ArcheOpterix [Ale+09] is a framework for the optimization of deployment architectures in embedded systems. The models have to be specified in the Architecture Analysis and Design Language (AADL). Due to the use of genetic algorithms, it is extendable to all quality attributes that are interesting for the domain of deployment architectures, e.g., safety, reliability, security, performance, timeliness, and resource consumption, among others.

PerOpteryx [KR11; Koz14] is another approach for component-based software architecture optimization. It supports the import of instances of the Palladio Component Model (PCM), but model-transformations, e.g., to Layered Queuing Networks, can be applied to use particular simulators or analyzers. A prototype supports the quality attributes performance, reliability, and cost. However, PerOpteryx can be equipped with new

quality attributes, analyzers, and degrees of freedom, as it uses genetic algorithms. For example, Yasaweerasinghelage et al. [Yas+19] extended PCM and PerOpteryx in order to add support for security constraints. Domain-specific rules, so-called tactics, are used to speed up the search.

ArchE [Bac+05; DP+08] is a reasoning framework that assists the architect in finding trade-offs between different architectural candidates. The knowledge about each quality attribute is encapsulated inside of a modular unit. The optimization is carried out through the application of rules, which are proposed to the architect. Thus, the architect needs to participate in the optimization process. DesignBots [DPC08] is similar to ArchE, but the modularization is even more sophisticated and the proposed local changes can be merged into a global change plan.

The SQuAT [Rag+17] approach reuses the concept of modular units, so-called bots. Each of these bots can analyze and optimize a given model with regard to one particular quality attribute for one scenario. The current prototypical implementation uses PCM for the specification of architectures and consists of two types of bots: one for performance and one for modifiability. Each of these bots performs a local search. Then, negotiation is applied to find meaningful trade-offs between these solutions.

2.2. Information Visualization

The visualization of abstract data is covered by the domain of information visualization. Works in this domain present concepts and technologies that can be adapted to visualize specific information. Therefore, this domain is also of relevance to this work. One important concept in information visualization is the mapping of data to visual variables. For this reason, Section 2.2.1 examines this concept in more detail. Additionally, various visualizations for different kinds of data have already been developed. Section 2.2.2 presents relevant examples for the visualization of multivariate data, while Section 2.2.3 focuses on examples of visualizations for relational data. In addition, Section 2.2.4 outlines how temporal data can be visualized. However, the domain of information visualization does not only cover visualizations, it also incorporates techniques for interaction with these visualizations. Hence, some of these techniques are described in Section 2.2.5.

2.2.1. Visual Variables

Understanding the concept of how to map data to visual variables is important to design or select suitable visualizations. The first step is to examine the type of data.

In general, four different types can be distinguished [Ste+46]. *Nominal-scaled* data allows distinguishing items based on their names to categorize them, e.g., by gender. *Ordinal-scaled* data also has the property that it can be ordered. However, the differences between the values might vary, e.g., rankings in a race or Likert Scales [Jos+15]. For *interval-scaled* data, also differences are defined, e.g., temperature in Celsius. If the data also has a meaningful zero, it allows for statements about the ratio of attributes. In this case, the data is categorized as *ratio-scaled*, e.g., temperature in Kelvin. The interval and ratio scales are also denoted as the cardinal scales.

So-called visual variables, e.g., position, size, shape, and density, carry information in visualizations [BBW83]. These variables have been investigated with regard to their suitability for perceptual tasks and data types, e.g., position is perceived much more accurate than density for quantitative data [Mac86]. According to Mackinley [Mac86], position, color hue, and texture are most suitable for nominal-scaled data. For ordinal-scaled data, position, density, and color saturation allow for high accuracy in solving perceptual tasks. For cardinal-scaled data, position, length, and angle are most suitable.

Works on perception focus on how humans perceive the world around them. Ware [War12] describes visual properties that can be rapidly detected in tasks on large data sets. Solving such tasks in less than $200 - 250$ milliseconds is possible due to preattentive processing. According to Ware [War12], visual properties that allow for preattentive processing exist for color, form, movement, and spatial positioning. Some examples for such visual properties are line orientation, line length, color hue, color intensity, flickering, and 2D-positioning. In addition, so-called Gestalt Laws [Wer23] describe the perception of patterns. These laws state that items are perceived as groups because of proximity, similarity, closure, connectedness, continuity, symmetry, common fate, and good form. These laws can be applied to create good designs and visualizations.

2.2.2. Visualizations for Multivariate Data

Many types of visualizations have already been proposed. Some of them are suited for data with more than two variables, so-called multivariate data. A selection of examples is presented in Figure 2.2.

Visualization (a) shows a scatter plot matrix [TT81] with three variables. This visualization combines several scatter plots, each capable of showing a pair of variables. The matrix shows all possible combinations of variable pairs and can, therefore, be seen as a projection into two-dimensional space. Each circle in each of the scatter plots belongs to the same data row. To allow for a better perception of particular data rows, the corresponding circles can be rendered in the same color, which results in so-called

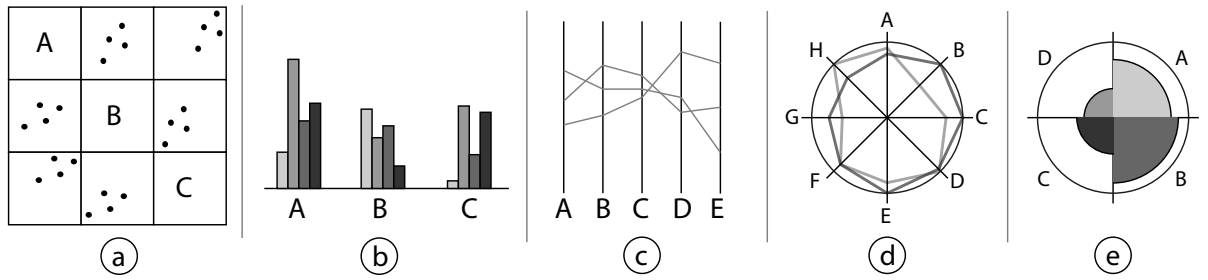

Figure 2.2.: Examples of multivariate data visualizations: (a) scatter plot matrix (b) bar chart (c) parallel coordinate plot (d) radar chart (e) petal diagram.

brushing scatter plots [BC87]. Many variations of scatter plots exist [CM84]. In one variation [Joh+83], symbols are used to show another variable inside of the scatter plots. It is also common to remove the bottom or upper triangle, as no information is lost in the process. A drawback of scatter plots matrices is their quadratic grow for increasing numbers of variables.

Visualization (b) is an example of a bar chart. Three variables and four data rows are visualized by grouping bars for each of the variables. However, other techniques for grouping [Jar89] and layouts [Mie14] exist, e.g., the bars can grow vertically or horizontally. It is also possible to connect the tops of the bars with lines to increase the visibility of the profile. Although the length of the bars allows for an accurate comparison, distant bars can be hard to compare for increasing numbers of variables and data rows.

High-dimensional visual analytics, like proposed by Wilkinson et al. [WAG06], is also a particular research field with special significance for this work, as the results for multiple scenarios or quality attributes can be visualized in a single visualization. An example for visualizations of high-dimensional data are parallel coordinate plots [Ins85], as shown in visualization (c). One vertical axis for each variable is shown. Each data row is represented as a polyline that intersects the vertical axes according to its values. Sophisticated techniques, like angular brushing [HLD02], have been developed to further improve these visualizations and allow for a better perception of particular properties in a data set. An advantage of parallel coordinate plots is that they can show patterns between nearby axes, but the axes are also order-sensitive and interaction is required to follow particular polylines in huge datasets.

Multiple data values can also be shown in a graphical object, a so-called glyph [War12]. Ward [War02] emphasizes that they are "especially well suited for displaying complex, multivariate data sets". He also mentions one particular type of glyph: the star plot [War02], which is similar to Parallel Coordinate Plots, but more compact. A star plot with eight variables and two data rows is also shown in visualization (d). Star plots

are also denoted as radar charts or spider-web charts because the lines form shapes similar to spider webs. This allows for the quick perception of a data row, but many data rows should usually not be displayed in the same chart due to the compact design. The petal diagrams [TF98], as shown in visualization (e), are similar to the previously examined star plots. However, they do not map the variables to the axes, but the regions between the axes. The perception of the shape is not order-sensitive, as the size of the region stays the same for a changed ordering.

Numerous other techniques and types of visualizations have been proposed. For example, variables can be mapped to segments of boxes [KH81], arrows [Eve78], or stick-figures [PG88]. Another approach maps variables to the corners of so-called harmonious houses [Kor91]. Chernoff's faces [Che73] make use of humans' perception of facial expressions by mapping variables to them. There is also the possibility to map high-dimensional data to lower dimensions, e.g., by a principal component analysis [WEG87]. Besides, Fourier transformations can be used for the same purpose, as in Andrews' plots [And72].

There are not only techniques but also sophisticated tools that help the user in the analysis of multi-dimensional data sets. One example is the Trade Space Visualizer [Stu+04], which focuses on the discovery of relationships between information. In addition, it also supports features like uncertainty and derivative analysis. Such tools help to answer general questions but are not specifically designed to answer specific questions, e.g., related to software architecture optimization and software models.

2.2.3. Visualizations for Relational Data

Graph-based visualizations are suited for relational data and hierarchies. Visualizations in this category all have in common that their underlying data structure is a graph [Wes+96]. Graphs consist of vertices, which are connected by edges. Additionally, the edges can be directed and weighted. Figure 2.3 shows three examples for graph-based visualizations.

Visualization (a) shows a typical graph-based representation, the node-link diagram [Sak+14]. In this diagram, the vertices are mapped to nodes, which are represented by circles. In general, other shapes could be used as well, e.g., rectangles. The edges are mapped to links, which usually have arrows at the ends to indicate the direction of the relation. In the example, the nodes *B*, *C*, and *D* have directed relations to node *A*.

A common problem of node-link diagrams is that they are vulnerable to visual clutter if many nodes are present. Visual clutter can be defined as "the state in which excess

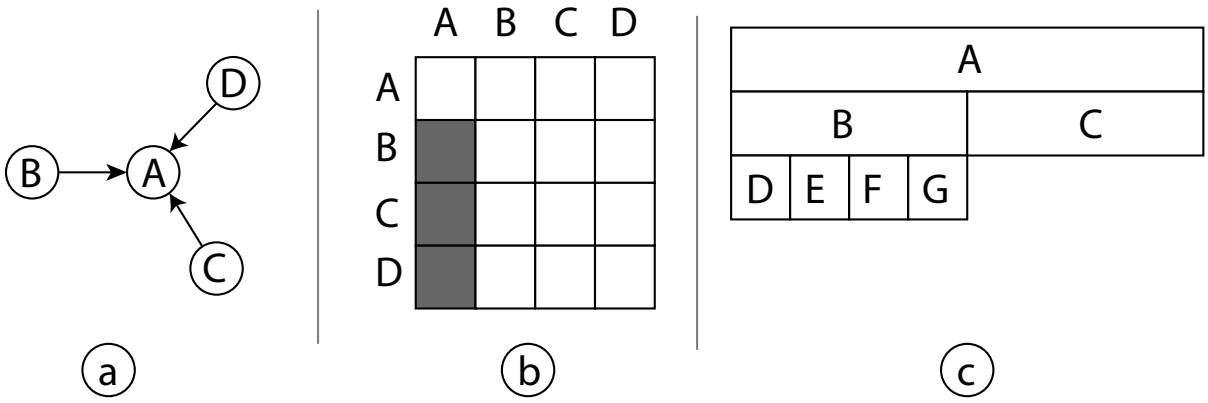

Figure 2.3.: Examples of graph-based visualizations: (a) node-link diagram (b) matrix (c) icicle plot.

items, or their representation or organization, lead to a degradation of performance at some task" [RLN07]. Visual clutter in node-link diagrams occurs because links and nodes can overlap. Therefore, layouts should be used to improve the representation.

In force-directed layouts [FR91], forces between the nodes and links are simulated. The result is a better-organized layout with uniform edge length and uniform distribution of nodes, in the ideal case. Therefore, force-directed layouts can mitigate visual clutter for up to 50-100 vertices and show symmetries. However, this technique requires a lot of computational power. Circular layouts [GK06] position the nodes in a ring. They are usually much faster and avoid to highlight nodes by their position. Hierarchical layouts [San95] position the nodes in hierarchical layers and are suited for (almost) hierarchical data. Another technique to reduce visual clutter is to merge edges, so-called edge bundling [Hol06].

A representation that avoids visual clutter is the matrix [Fek09]. Visualization (b) shows an example with the same setting as in visualization (a). Each cell in the matrix represents a possible link between two nodes. The source vertices are mapped to rows, while the target vertices are mapped to columns. In the example, a cell is colored black if an edge exists. This visualization gives a good overview, also for large numbers of vertices. However, path related tasks are more difficult to solve in this visualization.

Trees are special graphs. They have a unique root element, are directed, and are acyclic. Due to these properties, special visualizations for hierarchical data have been proposed. Visualization (c) shows an example of such a representation, the icicle plot [KL83]. In icicle plots, the root is at the top and the leaves at the bottom of the visualization. Vertices are positioned below their parents. In the example, *B* and *C* are children of *A*. *D*, *E*, *F*, and *G* are children of *B*. Coloring and width of the rectangles are sometimes used to display other variables in the same visualization.

2.2.4. Visualizations for Temporal Data

Another interesting domain for this work is time-oriented visualization [Aig+11]. Such visualizations are especially suitable to show how information evolves in time. Time can be treated like every other variable. For example, it can be placed on one axis of a scatter plot to show how the other variable behaves over time. Points in scatter plots can also be connected by arrows to show how two variables changed over time. It is also possible to place multiple visualizations for different points in time next to each other. Such visualizations are called comic strips [BC99]. A similar concept is the use of a time-sliders to modify the state of the displayed data with regard to time, as done in a work by Ahn et al. [Ahn+11]. Besides, 115 techniques for time-oriented visualizations are currently presented in the TimeViz Browser [TA15].

2.2.5. Interaction

According to Dix et al. [Dix+97], interaction is "the communication between users and the system". It is also an important concept to allow the user to get more accurate information or to comprehend the meaning of this information. Therefore, numerous techniques for different types of visualizations have been proposed.

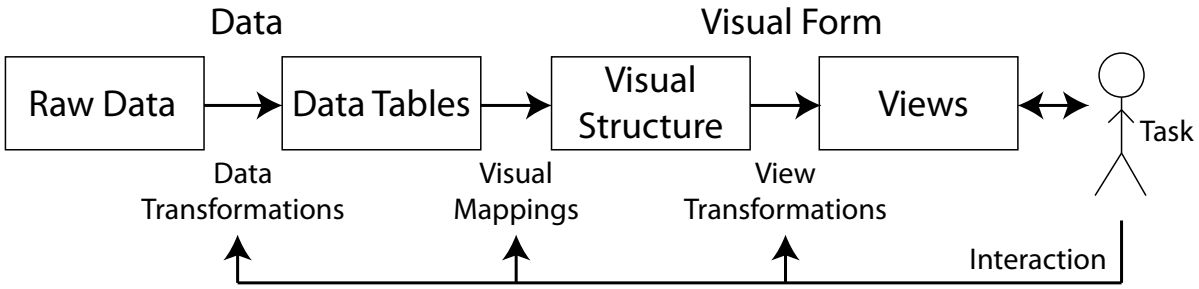

Figure 2.4.: Information Visualization Reference Model [Car99]

A more general description of interaction is given by the Information Visualization Reference Model [Car99] as presented in Figure 2.4. In this model, interaction can influence three parts of a system. Firstly, it can change the data transformation that is applied to the source data. For instance, data can be filtered, aggregated, or extracted. Secondly, the mapping from data attributes to visual variables can be changed by the user. Finally, the user can decide how to render the visual data and change the visible views.

Shneiderman [Shn96] gives the general advice to provide an overview first, then zooming and filtering should be applied, and details should only be delivered on demand.

Therefore, zooming [VWN03] is a commonly used technique to design visualizations that provide both: an overview and details. Another common interaction technique is brushing. It describes the highlighting of data items, e.g., by color, and is well suited for the coordination of multiple views [Rob07; WBWK00].

Chapter 3

Related Work

This chapter outlines the relevant works related to this thesis. Section 3.1 focuses on the current state of visualization support in the domain of software architecture optimization. Since there are no works that explicitly try to tackle the same problem as this work, also works from domains related to software architecture optimization are examined. Therefore, works that visualize similar data or use visualizations to solve similar problems are considered.

Generic visualization approaches are rather designed to visualize specific data types than making restrictions on specific tasks. Thus, some of these approaches outlined in Section 3.2. Section 3.3 describes works from the domain of interactive optimization because some software architecture optimization approaches also try to overcome their limitations by the participation of humans in the optimization process. Thus, the two domains are related to each other. Finally, Section 3.4 focuses on works related to the visualization of software architectures or their properties, since they are the subject of optimization in software architecture optimization.

3.1. Visualization in Software Architecture Optimization

Multiple state-of-the-art software architecture optimization approaches have already been presented in Section 2.1.4. While all of these approaches focus on the functional part of software architecture optimization, they only offer no or limited support for the architect's task to evaluate the solutions. Some of the mentioned approaches provide result visualizations, e.g., ArchE [Bac+05; DP+08] uses traffic light glyphs to show whether a scenario is/can be fulfilled or not. However, they do not provide an overview of a large design space and hide the actual values. In the works on AQOSA [Li+11]

and ArcheOpterix [Ale+09], scatter plots are utilized to show the Pareto front, but this visualization is only suitable for low-dimensional data.

Approaches which are specifically designed to support interactive optimization, for instance, DesignBots [DPC08] and ArchE [Bac+05; DP+08], at least provide basic graphical user interfaces to support interaction. However, other approaches, e.g., SQuAT [Rag+17] and PerOpteryx [KR11], provide results only in textual form. Then, it is the responsibility of the architect to find tools that support her in the visual assessment of the results.

Although there are no visualization approaches specifically focusing on software architecture optimization, some visualization approaches for their optimization strategies exist. Visualizations for genetic algorithms are used, for example, in the tool GAVEL [HR01] to allow for visual comprehension of the optimization process. Graphs, bar charts, and line graphs are used to display the history of chromosomes and the usage of applied operations, e.g., crossover or mutation. However, these works focus too much on specifics of genetic algorithms to be of high relevance for this work. Additionally, not all software architecture optimization approaches use genetic algorithms.

3.2. Generic Visualization Approaches

Generic visualization approaches rather focus on the visualization of specific data types than on assisting users in domain-specific tasks. State-of-the-art software architecture optimization approaches often output multivariate data because there are often more than two goals defined. Several tools for the visualization and analysis of multivariate data exist. The Trade Space Visualizer [Stu+04], for example, aims to help in discovering relationships between variables. Therefore, it can generate scatter plots, histogram plots, and parallel coordinate plots, among others. In addition, it supports analysis of the data, e.g., principal component analysis [WEG87], and provides interaction techniques, e.g., brushing. Another example is ClustVis [MV15], a web tool for performing principal component analysis [WEG87] and creating heatmaps from the given data. Multiple methods for the visualization of multivariate data are also used by the Xmdvtool [War94]. It visualizes data as star glyphs and parallel coordinate plots. Besides, it also allows for interactive examination of this data.

However, these tools are not suited for the visualization of software architectures. This means that important information in software architecture optimization can not be visualized properly. In addition, the data provided by software architecture optimization approaches needs to be transformed to fit the input format of the visualization approaches. In general, there is also no possibility to export data in such a way that it

could be used by interactive software architecture optimization approaches. Therefore, using them is often inconvenient for software architects.

3.3. Visualization in Interactive Optimization

Interactive optimization faces the complexity of optimization problems by using human knowledge in the optimization process. It combines knowledge and methods from the domains of optimization, visualization, and human-computer interaction. Therefore, it has many similarities to this work but does not focus specifically on the domain of software architecture optimization and its characteristics.

Meignan et al. [Mei+15] provide a taxonomy for the field of interactive optimization and also give an overview of the existing approaches. Five main categories are described: *trial and error* approaches, *interactive reoptimization* approaches, *interactive multiobjective optimization*, *interactive evolutionary algorithms*, and *human-guided search*. *Trial and error* approaches do not generalize the user feedback, so each iteration is independent. These approaches are used to adjust the optimization problem or to tune the optimization procedure. An example is an approach [Ces+03] that finds schedules for the transmission of data by a space probe. *Interactive reoptimization* aims to refine optimization problems. Solutions are adjusted by a reoptimization procedure if the user makes (local) modifications. An example is an approach [VBRK92] that searches for solutions to vehicle routing problems. *Interactive multiobjective optimization* focuses on problems with different, conflicting objectives. The user provides her preferences in the decision-making process, which can result in better solutions. The NIMBUS method [MM00] is such an approach, in which the user assigns classes to objective functions in order to express her preferences. *Interactive evolutionary algorithms* partially or completely rely on the users when it comes to the evaluation of solutions. This method is usually applied if the quantification of objectives is difficult, e.g., for the evaluation of fashion designs [KC00]. Finally, *human-guided search* can improve the efficiency of optimization procedures. Heuristic information provided by users guide the optimization process, even without a direct impact on the optimization model. An example of a *human-guided search* is the platform by Klau et al. [Kla+02].

Several examples of interactive optimization approaches exist. However, many are linked to specific problems. For example, the tool Dunnart [DMW08] helps the user to find a suitable layout for a graph in an interactive way. Some methods in this field also make strong mathematical assumptions, e.g., differentiability [Bra+08].

There are also works focusing on visualizations for interactive optimization approaches. Jones [Jon94] presents techniques that are suitable for displaying optimization models,

algorithms, and solutions in interactive optimization. He discusses bar charts, pie charts, graph-based, and matrix-based visualizations. Miettinen [Mie14] discusses several types of visualizations for displaying solutions to decision-making problems, e.g., bar charts, star plots, petal diagrams, and metroglyphs, among others. While none of them is evaluated to be better than the others in general, it is emphasized that some might be more suitable for particular situations. Thus, these visualizations are compared and their advantages and disadvantages are emphasized. The visualizations techniques are investigated with regard to the possible number of criteria that can be visualized, order sensitivity, the potential number of alternatives to be displayed, and whether ranges are needed. In addition, choosing the right visualization type can also depend on the users' way of thinking and not only on the problem to solve. It is, therefore, suggested to offer the decision-maker the choice of selecting appropriate visualizations.

There are also numerous approaches and tools in interactive optimization which use visualization techniques. For example, the multi-objective optimization approach WWW-NIMBUS [MM00] uses bar charts, value paths, and petal diagrams to visualize solutions. However, as the domain of interactive optimization does not require to optimize software architectures, the concepts can not simply be transferred to the more specific domain of software architecture optimization. Therefore, works that help to select suitable visualization techniques, like the one by Miettinen [Mie14], are more useful than particular approaches.

3.4. Visualizations for Software Architectures

Architects in software architecture optimization should be able to select the best architecture for their purpose or further search. They also want to comprehend the made changes. In principle, architects could use graphical modeling editors to asses the candidates, but these editors are usually not designed to answer these questions efficiently. Still, they are often the first choice of architects in software architecture optimization due to a lack of suitable alternatives.

Modeling languages can also be used to model software architectures. These languages must have at least one textual or graphical syntax. For example, the Unified Modelling Language (UML) [RBJ17] suggests graphical notations for various diagrams, which describe the static and dynamic behavior of software systems. The Palladio Component Model (PCM) [Reu+11] even comes with its own graphical model editor, which enables the architect to model components, the system, the hardware environment, and the users' usage behavior. The Open Source AADL Tool Environment (OSATE) [Fei04] also provides a graphical model editor.

However, the purpose of these editors is to enable the architect to model all supported properties of a particular software system. Thus, their visualizations are detailed and not designed for comparisons. Most representations also seem to be graph-based. This amplifies the problem of information overload for the architect, especially for big models, as graphs are susceptible to become visually cluttered.

SiLift [Keh+12] can be used to compare EMF models [Ste+08]. It automatically derives the necessary editing operations from basic transformation rules. These editing operations are then visualized in a compare view. SiLift is based on the rule-based model transformation engine EMF Henshin [Str+17]. Henshin also provides a visual syntax, which maps colors to different kinds of actions, e.g., added elements are colored green. Thus, it is possible to identify differences between models and to visualize these differences.

However, there are also approaches that are not directly related to particular modeling languages. Galagher et al. [GHM08] describe a framework for the evaluation of software architecture visualization approaches. Six of these approaches are evaluated with the framework. The evaluation also contains categories for model comparisons and change visualizations. It can be concluded that the investigated approaches only partially support comparisons. An exception is LePUS [Ede02], but it is a modeling language, which is specifically designed to enable visualization of programs. Thus, it is not a visualization tool. In addition, all the investigated approaches do not support showing rationale or evolution.

Another example from the software architecture visualization domain is SAVE (Software Architecture Visualization and Evaluation) [DKL09], which supports structural and behavioral views. Although the basic idea is to support comparison, it is not the comparison between multiple model instances, but between the implemented code and the specification. In addition, SAVE is also able to show a similarity metric for modules to support comparisons between software variants.

Salameh and Aljammal [SAA16] conducted a literature survey on techniques for software evolution visualizations. They conclude that software repositories are the sources of information for most of the approaches. In addition, most of the approaches use graph-based or matrix-based, only a few notation-based or metaphor-based visualization techniques. The focus of most of the approaches lies on the architecture or the artifacts (in the repository), instead of metrics or features. For example, one of the investigated approaches [LL07] focuses on inter-module dependencies and relations. They propose the “Semantic Dependency Matrix” and the “Edge Evolution Filmstrip” as visualizations for this purpose.

There are also software visualization approaches that focus on quality attributes and metrics. For example, E-Quality [ETB11] is a tool for object-oriented software quality

visualization. It is able to extract quality metrics and represents them in graph-based, interactive visualizations. In contrast to software architecture optimization, the focus lies on the detection of design flaws in one particular architecture.

Another approach by Langelier et al. [LSP05] visualizes classes as colored three-dimensional objects inside of treemaps and starbursts. This approach aims at supporting the architect in analyzing software systems of a large scale. However, the approach can not display multiple instances or versions. The latter one is stated as planned and sliders are mentioned as one possibility for the implementation.

A work by Goodwin et al. [Goo+17] investigates the requirements of users from the domain of constraint programming on visualizations. There are some similarities to software architecture optimization, which also involves models and search, but this work focuses on the profiling of performance issues. The users emphasize the need for visualization for formalizing problems, result representation, and debugging. Tree-like visualizations are used to represent the search, namely radials trees, icicle plots, and sunburst diagrams. There are also combinations of timelines and search trees, which are evaluated as useful by the users. In this evaluation, the users perceive the icicle plots as the most useful, examined alternative tree representation.

Chapter 4

Concept

Creating an approach that can be used to assist architects in the domain of software architecture optimization is a complex task. Due to the fact that such an approach relies on results provided by existing software architecture optimization approaches, it is necessary to examine the characteristics of these approaches and their outputs. For this reason, Section 4.1 outlines the basic commonalities and differences of existing software architecture optimization approaches. The next step is to identify the typical challenges that an architect encounters in this domain. Therefore, Section 4.2 presents the functional requirements based on typical use cases in the domain. Due to the differences between existing software architecture optimization approaches, it is hard to create an approach that fits all of these existing approaches. Design decisions have to be made to determine how the approach should handle, or not handle, these differences. These conceptual decisions are described in Section 4.3.

Based on the requirements and the design decisions, the actual approach can be designed. Therefore, Section 4.4 summarizes the most important concepts that are applied to the approach as a whole. To visualize the results of software architecture optimization approaches, suitable visualizations have to be chosen. Section 4.5 elaborates on these choices. Besides, it mentions modifications and interaction techniques to supplement the visualizations and improve their usefulness for the specified tasks. These visualizations are assembled in so-called views, as described in Section 4.6. In these views, different visualizations are displayed together and architects can manipulate them through interaction. Thus, the presented views form the core of the visualization approach.

4.1. Domain Analysis

To determine the data-structure for the approach, it is necessary to identify the commonalities and differences between the existing software architecture optimization approaches. A simplified, generic model for the domain of software architecture optimization approaches is therefore shown in Figure 4.1.

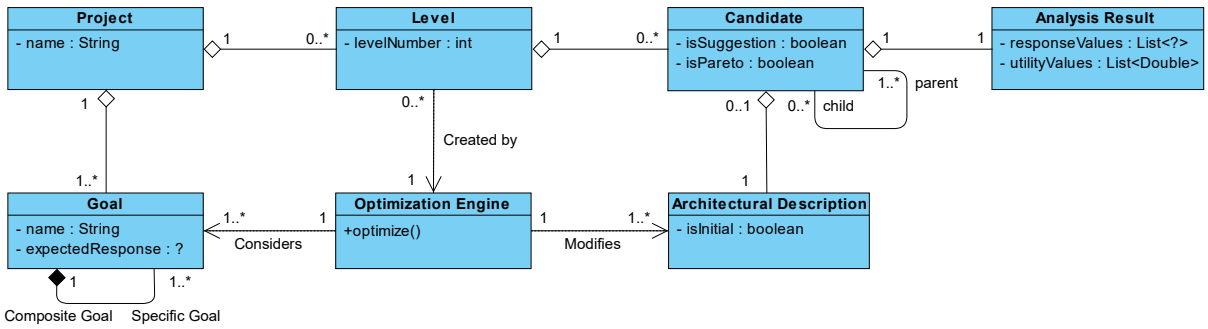

Figure 4.1.: Domain model for software architecture optimization.

This model has been designed based on practical experience from the development of SQuAT [Rag+17] and the usage of PerOpteryx [KR11]. It has been generalized to also fit to other approaches, for instance AQOSA [Li+11], ArcheOpterix [Ale+09], ArchE [Bac+05; DP+08], and DesignBots [DPC08]. The purpose of this domain analysis is also to provide a visualization approach that can be connected to most of the existing software architecture optimization approaches. It has to be mentioned that not all approaches expose all these features to the architect, but most of them are least supported in the approach's concept. Thus, it would be possible to support them with minimal effort.

- **Project:** The project is the container for the whole dataset, which is created within a single run of the optimization engine. It also contains the parameters for this run, e.g., the goals.
- **Optimization Engine:** The optimization engine is the heart of every software architecture optimization approach. It modifies architectures to reach specified goals. The result is a set of candidates, which meets the stated objective(s), in the ideal case. Therefore, the optimization engine must be able to evaluate architectures with regard to the specified goals. It also needs to be equipped with transformations to modify the initial architecture(s). It contains almost the whole optimization process, which is not of interest for the visualization approach. Therefore, this process is abstracted in a single class.

- **Goal:** Optimizing a software system always requires to specify goals for the optimization. If the architect can not specify the goal, then it is implicitly given by the approach. However, the exact definition of goals can vary from approach to approach. SQuAT [Rag+17] uses so-called scenarios [CKK02]. These scenarios consist of a stimulus (change) to the system, and an expected response measure when applying this stimulus. In PerOpteryx [KR11], the top-level goals are quality attributes, e.g., performance, and metrics are the atomic goals, e.g., the system's response time. Thus, only the atomic goals are actually considered in the analysis. The relevant difference, however, is that SQuAT accepts an expected response, while PerOpteryx tries to minimize or maximize a metric. Apart from this difference, the biggest challenge is the data type of the metric, e.g., a comparison of response times in seconds to reliability in percent is not possible. Even the direction of improvement is different, because response time gets better for smaller values, while reliability needs to be maximized.
- **Analysis Results:** The analysis result describes the degree to which a candidate fulfills the specified goals. For each of the goals, a response value has to be determined. In order to compare the goals with each other, approaches like SQuAT [Rag+17] or ArchE [Bac+05; DP+08] also map the response values to utility values. A value of 0 means that the goal is not fulfilled, while a value of 1 means that the goal is fulfilled. Thus, these utility values represent the architects' preferences in a comparable way.
- **Architectural Description:** All the approaches rely on an underlying architectural model to represent the architecture. However, there are differences between the approaches, e.g., PerOpteryx [KR11] relies on PCM [BKR07] to represent the architecture, ArcheOpterix [Ale+09] uses AADL [FG12], and AQOSA [Li+11] also supports ROBOCOP [BC+06]. Nevertheless, there is also a commonality between the approaches: They all take at least one initial architecture as input. This is reasonable because these approaches can only optimize something that has been specified before.
- **Candidate:** The candidate can be seen as the container for the results of the optimization. It connects the analysis result and the architectural description. Some approaches perform an additional analysis to suggest candidates to the architect. For example, PerOpteryx [KR11] and ArcheOpterix [Ale+09] identifies the Pareto candidates. While this seems to be a common and reasonable technique, also other kinds of suggestions exist, e.g., SQuAT [Rag+17] applies negotiation techniques to provide suggestions.
- **Level:** A level consists of candidates that are generated together. This means that there is level that contains all the parental candidates of another level. PerOpteryx [KR11], for instance, internally creates levels due to its underlying evo-

lutionary algorithm. Interactive approaches, like ArchE [Bac+05; DP+08], ask the user to select candidates (see Section 4.2.2) after each level. Approaches without such a structure provide only two levels: the first level contains the initial candidate(s) and the second all the proposed candidates.

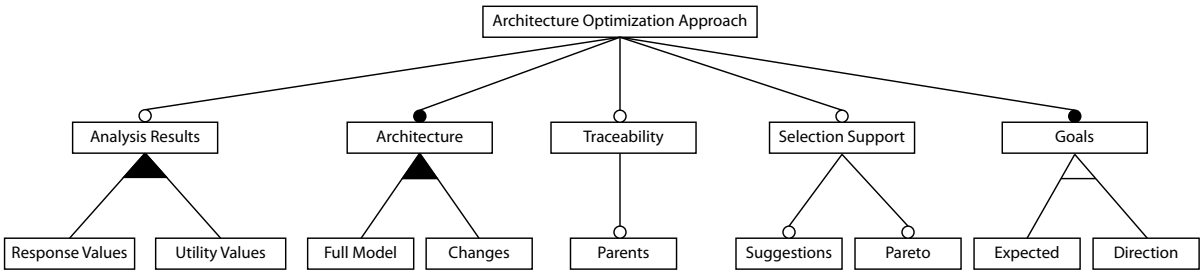

Figure 4.2.: Configurations of software architecture optimization approaches and their results.

As mentioned before, not all the approaches share all the existing information with the architect. This is important because the output of the software architecture optimization approaches is the input for the visualization approach presented in this work. Thus, Figure 4.2 outlines the most important, possible differences between the approaches.

Usually, an approach provides both, architectural descriptions and analysis results. However, an architectural description is necessary for the architect to implement the optimized system. The approach can either return a full specification of the candidate's architecture, like in SQuAT [Rag+17], or provide only the differences to the initial candidate, like in PerOpteryx [KR11]. If the analysis results are provided, then it is easier to see the strengths and weaknesses of the proposed candidates. As already explained, response values and/or utility values can be returned. With regard to traceability, an approach can decide to inform the architect about the parent-child-relationships between the candidates. For example, ArchE [Bac+05; DP+08] does this implicitly in the interactive process. As explained above, software architecture optimization approaches can provide support for the architect in finding the "best" candidates. Whether this is a Pareto analysis [NZES05] or another technique, this is an optional feature. Goals, on the other hand, are mandatory. Some approaches require the architect to state her expectation, e.g., SQuAT [Rag+17], while others (implicitly) use an optimization direction, e.g., PerOpteryx [KR11].

4.2. Functional Requirements

In software engineering, it is good practice to state the requirements of a system before implementing it. This is necessary to validate the software system at the end of the development process. However, for the design of a visualization approach to support architects using software architecture optimization tools, exact requirements are hard to define. One reason is that there is often not an objectively "correct" solution. Especially in such cases, correctness is not the only important property of a system. Trust and confidence in the solution play an important role as well [LS04]. The use of interactive methods is a way to foster this trust. Therefore, this work rather aims at assisting architects in their natural, subjective behavior, which is described by their generic use cases.

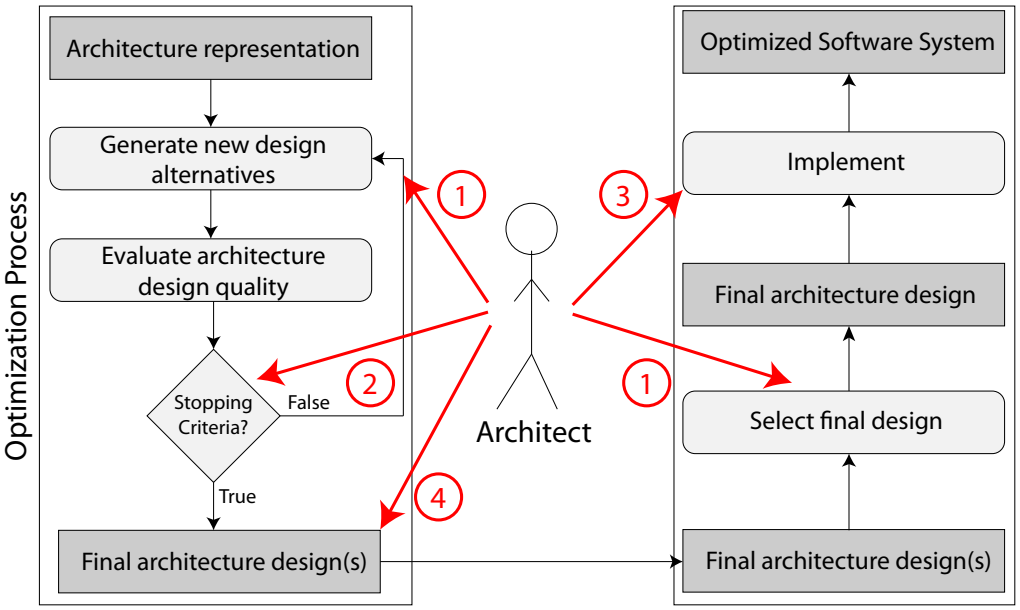

Figure 4.3.: Use cases for architects in software architecture optimization. The red arrows show which part of the software architecture optimization process is affected. The red numbers refer to the use cases *UC1*-*UC4* in this work. Optimization process adapted from Aleti et al. [Ale+13].

The overall process of gathering the requirements is outlined in Section 4.2.1. Then, the Sections 4.2.2-4.2.5 describe the relevant top-level use cases. Figure 4.3 provides an overview of the described use cases. In addition to the description, the reasons for defining these use cases and their impact on this work are described. Finally, the relevant properties of visualizations for the use case get derived. Table 4.1 gives an overview of the results. A visualization can either support the response values of goals or the architecture as displayed data. Comparisons of multiple values or architectures can give

Category	Subcategory	UC 1	UC 2	UC 3	UC 4
Data Type	value	● ● ●	● ● ●	○ ○ ○	● ● ●
	architecture	○ ○ ○	● ○ ○	● ● ●	● ● ●
Comparison	detailed	● ● ●	● ○ ○	● ● ●	● ○ ○
	overview	● ● ○	● ● ●	○ ○ ○	● ● ●
Interrelationship	candidate/goal	● ○ ○	○ ○ ○	○ ○ ○	● ● ●
	candidate/architecture	○ ○ ○	○ ○ ○	○ ○ ○	● ● ●
Evolution	parents	○ ○ ○	● ● ○	○ ○ ○	● ● ○
	changes	○ ○ ○	○ ○ ○	● ● ●	● ● ○

Table 4.1.: Requirements on visual properties for the use cases covered by the visualization approach. The importance of each property is expressed on a 0-3 scale, where 0 is *unimportant* and 3 *very important*. ○ has a value of 0, ● has a value of 1 and can be summed up.

an overview of the whole population, or be more suitable for a small subset and provide more details. While comparisons mean that the same property of a candidate, e.g., the architecture, is compared to the one of another candidate, interrelationships refer to dependencies between different data types. Visualizations that can reveal dependencies between candidates and goals in general, and between candidates and architectures are distinguished. In addition, it is investigated whether visualizations need to support evolutionary information. This includes the visualization of information about parental candidates and changes between parent and child candidates.

4.2.1. Methodology

In theory, it would be possible to ask the domain experts at the beginning of the projects for their requirements. In practice, however, it is hard for humans to imagine a system which does not exist, yet. Their mental picture of the system becomes more precise during the development of the approach. For this reason, a more flexible method is used in this work. First, the top-level use cases (see Section 4.2.2 to 4.2.5) are described as a guideline for the further process. Based on these use cases, the relevant properties for visualizations are derived. Additional requirements about the results to visualize are implicitly given by the state of the art in the domain, which is described in Section 4.1. Visualizations for the stated properties (see Section 4.5) are designed and composed into views (see Section 4.6). Then a paper prototype [Sny03] is created and presented to domain experts. Thus, their feedback on missing features and possible improvements can be considered in the process. In the next step, an early version of the system is presented to the domain experts to gather more requirements. Finally, the domain

experts use the system on their own and provide feedback. This process intends to include the domain experts early, so the approach can be designed to their needs. The results of the interviews with the domain experts are described in Chapter 6.

4.2.2. Use Case 1: Candidate Selection

The first use case covers situations in which an architect has to select a subset of candidates from a set of candidates. The subset should contain only the "best" candidates and is used as input for another process, e.g., another optimization run. The definition of "best" is left to the architect's discretion. Selecting candidates can have different reasons, for example, a software architecture optimization approach can include human knowledge into the optimization process. Thus, this use case is common in the domain of interactive optimization, as described in Section 3.3. In interactive optimization, the optimization is interrupted and the architect expresses her preferences for the next iteration in the optimization process. In the context of this use case, architects can select candidates which are, in their opinion, the "best" to serve as parental candidates for the next optimization steps.

Although this use case seems to arises mainly in the domain of interactive optimization, it also applies to non-interactive software architecture optimization approaches. At the end of the optimization, there are usually multiple candidates with strengths and weaknesses. While this set can be objectively reduced by a Pareto analysis [NZES05], still multiple candidates can remain at choice. The architect can then select the candidate, which serves her purpose best. Even for approaches that only provide a single result, the architect has to decide whether to accept or decline the solution. Thus, this use case applies to all types of software architecture optimization approaches.

Still, the definition of what "best" candidates are varies depending on these types of approaches and the preferences of the architect. For this reason, visualizations should be chosen in such a way that they show the essential information to the architects, who can then make a choice on their own. This can also embrace information that supports the decision-making process, e.g., by tagging the Pareto candidates.

The specified goals and constraints for the optimization can be seen as the initial expression of the candidate's preferences. Therefore, the focus of the visualizations should lie on the response values, which express the satisfaction of these goals. It is vital to be able to conduct detailed comparisons between candidates. An overview of the whole population can also be useful to initially reduce a big population to a smaller subset quickly. Additionally, It can help the architect to evaluate a single candidate with respect to the existing values in the population. For example, a candidate should not be excluded even if it does only partially fulfill the requirements, while most of the

candidates in the population perform even worse. In this case, the architect should investigate whether her stated goals are satisfiable.

4.2.3. Use Case 2: Stopping Criterion

This use case refers to the stopping criterion mentioned in Section 2.1.3. Architects have to decide whether the optimization process should be continued or terminated. Obviously, this use case becomes obsolete if a solution is already found that fully satisfies the architects' expectations. This question, however, is already investigated in the previous use case. Anyways, for architectures with many degrees of freedom and many goals and constraints, it might be better to make some trade-offs and accept a compromise. This is due to the fact that software architecture optimization itself is often a time- and resource-consuming process. Thus, the decision to stop or to continue is strongly influenced by economic considerations.

There are approaches that offer to define stopping criteria. For example, PerOpteryx [KR11] allows to define a maximum number of iterations, as well as a threshold for the necessary improvements after some iterations. The drawback of such automated solutions is that the architect has to define the values at the start of the optimization process. At this point, the architect usually does not have enough information for a reasonable decision, so she needs to guess. Thus, the decision often gets just postponed. At the end of the optimization, the architect still has to judge whether restarting the optimization process with other parameters would be beneficial. Thus, this use case again applies to interactive and non-interactive software architecture optimization approaches. The only difference is that the cost of restarting a non-interactive approach is often usually higher if there is no way to continue the terminated optimization process.

As in the previous use case, response values play an important role in the decision. Still, there can be situations in which architectures are of interest to the architect, e.g., she discovers that promising changes to the architectures have not been investigated, yet. In general, this decision does rather require a good overview of the whole population than a detailed comparison of a small subset of candidates. Different levels of candidates can be compared to each other to see whether improvements have been made. This is a common approach and also implemented in PerOpteryx [KR11]. Note that this requires parental candidates to be visible.

4.2.4. Use Case 3: Candidate Implementation

The usual motive of architects in using software architecture optimization is to find a software architecture that meets their requirements. This means that, at the end of

the optimization process, the work for the architect is not finished. The architect still needs to know how to implement an accepted architecture. The software architecture optimization approaches usually do not consider this as part of their scope, because the model can be investigated in a model editor. However, an approach to visualize the results of software architecture optimization should not completely ignore this basic use case. It also has to be noted that investigating big models in model editors or on the textual level can be a time-consuming and error-prone task. In addition, software architecture optimization approaches are often able to provide the applied changes. It is, therefore, reasonable to assist architects in answering these questions by visualizing these changes. At least, architects should get an overview of the most important modifications.

When it comes to this use case, the response values become irrelevant. They served their purpose in the use case to select the candidate (*UC1*) and to terminate the optimization (*UC2*). For the implementation, the architecture is the relevant type of information to display. The straightforward approach is to visualize the changes that have been made during the optimization process. However, this requires the software architecture optimization approach to store and provide this information, which can not be guaranteed. A more general solution is to treat this as a special case of an architectural two-candidates comparison. One of these two candidates is the initial candidate that describes the existing system, while the other candidate is the one that should be implemented.

4.2.5. Use Case 4: Explain Results

Software architecture optimization approaches are often complex software systems and use sophisticated techniques. From the ordinary architect's perspective, most of these approaches seem to be black boxes that magically transform candidates and present them as the problem's solution. However, trust in the software system is important to accept the solutions [LS04]. In interactive approaches, the architects themselves make the decisions, which makes the optimization process more comprehensible. With regard to an approach that visualizes the outputs of software architecture optimization, trust can also be created by explainable results. Thus, the architect must be able to see more than just the individual properties of each candidate. She needs to see the interrelationships between these properties, e.g., conflicting goals can prevent that acceptable solution are found. Apart from the fostered trust, such insights are the foundation for well-justified actions to handle unsatisfying results.

For visualizations, this means that different types of data have to be displayed. Thus, also response values and architectures need to be displayed in the same visualizations or views. Detailed comparison of candidates might be useful to explain the results for

single candidates. However, assumptions based on the multiple candidates are more meaningful. Hence, an overview of the population is more important. Based on the overview, it can be determined how candidates in general behave for specific goals or changes in the architecture. Also, information about the evolution of candidates can provide insights as well, e.g., most of the accepted candidates can have the same parent. The architectural change between a parent candidate and its child can also help to explain why it performs better or worse.

4.3. Design Decisions

Section 4.1 highlighted some relevant differences between existing software architecture optimization approaches. Therefore, it has to be decided on how these differences in a visualization approach should be handled. On the one hand, supporting all features would result in high compatibility with all software architecture optimization approaches. On the other hand, supporting many features can significantly increase the complexity, and therefore, the maintenance costs of the visualization approach. Thus, this section describes how this work tries to overcome the issue of incompatibility between the different approaches. Section 4.3.1 focuses on the analysis results, while Section 4.3.2 describes how different architectural descriptions are handled. Then, Section 4.3.3 about selection support is followed up by Section 4.3.4 about traceability information. Finally, Section 4.3.5 outlines how goals are handled.

4.3.1. Analysis Results

As already explained, the response values are difficult to compare. There can be different data types, optimization directions, and measurement units, e.g., predictions can be reported in seconds or minutes, among many others. The concept of utility values is easier to handle. A utility function maps the response values to real-values in a way that they are ordered by preference of the response value [Deb54], e.g., 0 represents the lowest preference, and 1 represents the highest preference.

It has to be noted that not all approaches use utility values internally, but defining a function is easier for the architect or the software architecture optimization approach than for the visualization approach. The reason for this is that the architect knows her preferences and the software architecture optimization approach knows the semantics of the specified goal. A generic visualization tool, however, does not necessarily have this kind of information. Thus, the responsibility for delivering utility values is moved to the client-side, at least for the initial proposal of the visualization approach. However,

there is still the option to make the utility function a part of the visualization approach in the future. Then, the visualization approach could even allow the architect to change the utility functions during the analysis of the data.

4.3.2. Architectural Description

Different modeling languages are used by the investigated approaches and in the domain of software architecture optimization [Ale+13] as well. Some examples are AADL [FG12], PCM [Reu+11], and UML [RBJ17]. To be able to support all these languages, two different approaches can be applied. In the first one, one of these modeling languages, e.g., PCM, is selected as the underlying modeling language. To support other modeling languages, transformations between these models are necessary. However, due to the fact that these models support many features, defining these transformations would not be trivial. Supporting most of the existing modeling language would even mean that many of these transformations are necessary. Therefore, the other option is chosen: a new, minimal architectural model is designed. This model only supports a subset of the features that are provided by (most of) the common modeling languages for software architecture optimization. Although some detail is lost in this process, the necessary transformations are much simpler. It is also important to mention that the detailed information can still be visualized by visual modeling editors, which are actually designed for this task. For this work, however, a generic, and simple representation is better suited.

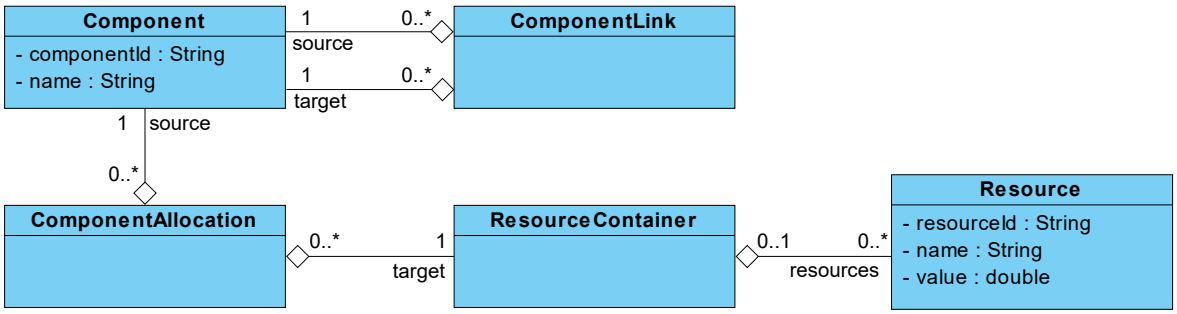

Figure 4.4.: Architectural model used in the visualization approach. The diagram shows the data types in the model.

As all of the mentioned modeling languages support the modeling of software and hardware properties, it is reasonable to adopt this. For component-based software, components and their dependencies are the most basic elements. For hardware, there are resources, like central processing units (CPUs) or hard disk drives (HDDs), which can

be part of a container. In practice, servers are such kinds of resource containers. Finally, the allocation of components to these servers describe how software and hardware interact. Figure 4.4 visualizes this minimal model.

According to Aleti et al. [Ale+13], the six most common degrees of freedom in software architecture optimization are *allocation*, *hardware replication*, *hardware selection*, *software replication*, *scheduling*, and *component selection*. Apart from *scheduling*, this simple model considers all of these degrees of freedom. Thus, it is likely that an architect is able to discover the changes in visualizations of the architecture.

4.3.3. Selection Support

It seems to be a common technique for many software architecture optimization approaches to apply a Pareto analysis. As this is a dispassionate technique to call the architect's attention to preferable candidates, it has been decided to make the Pareto analysis a part of the visualization approach. Thus, regardless of whether the software architecture optimization approach conducts a Pareto analysis or not, it will be conducted in the visualization tool based on the provided input. This assures that architects can identify interesting candidates easier, especially in big populations.

If an architecture optimization approach uses other techniques to suggest candidates, then this information should be displayed, too. The architect can still decide to reject the suggestions but can find interesting candidates more quickly. Therefore, this information can help the architect in selecting candidates, which is a use case in this work as described in Section 4.2.2.

4.3.4. Traceability

Traceability is an important property of the results if evolution should be visualized. In this work, this is the case for two use cases, namely deciding whether to stop the optimization and explaining results (see Section 4.2.3 and Section 4.2.5). For this reason, it is reasonable to support the linking of the parental candidate, if the information is provided. However, to avoid additional complexity, only single parents are supported. This assures that the number of parents in the trace back to the initial candidate is kept small. Hence, parents in the visualizations can be handled easier. It has to be noted that there can be multiple parents in practice, e.g., crossover in evolutionary algorithms creates a new candidate out of two existing ones. In this case, one of these two candidates has to be appointed as the parental candidate. Support for multiple parental candidates, however, can still be added in the future.

4.3.5. Goals

The most relevant difference between the definitions of goals seems to be whether they optimization in a direction or require an expected response. However, this difference becomes almost irrelevant if utility values are used. Then, a value of one is the desired response and higher values are better, regardless of the definition of the goal. What remains is the conceptual description of the goal itself, but this is something that is known to the architect, who defined the goal. The visualization approach, on the other side, only needs to know the unique name of the goal, so the architect can identify it.

4.4. Important Concepts

This section describes the most important concepts that have been designed for the visualization approach presented in this work. All of these concepts are of global scope and applied consistently within the approach. Section 4.4.1 explains how candidates can be grouped for a more efficient use of the approach. In contrast to groups, tags are rather individual properties of candidates and explained in Section 4.4.2 in more detail. Section 4.4.3 describes colors and symbols that are used to highlight groups and tags. Then, Section 4.4.4 outlines important features to improve the usability of the approach. Finally, Section 4.4.5 elaborates on the concept of global access and control of states and options.

4.4.1. Groups

Candidates can be assigned to different groups, like levels naturally group candidates. When candidates are selected, as explained in Section 4.2.2, then they are basically assigned to a common group. Because of this, the name of this group is *selected*. The *selected* candidates are the ones that are sent back to the optimization tool in interactive optimization approaches or should be exported as final results.

However, sometimes an interesting candidate is found, but the architect might not be sure yet, whether it should be part of the *selected* group. For this reason, the *marked* group is proposed. This group should contain candidates that are marked for further investigation.

Two other groups are introduced as well: Interaction with visualizations puts the highlighted candidates into the *current* group. Hence, these are candidates which are at least of temporary interest to the architect. For this reason, this group is also highly volatile. The whole group can be cleared by interaction and new candidates are added

to it. The fourth group is *comparison*. This group is intended to be used for a small number of candidates, which should be compared to each other in detail.

It has to be noted that *marked* and *selected* exclude each other. On the other hand, memberships in *current* and *comparison* can be combined with all the others.

4.4.2. Tags

Tags are similar to groups, but rather describe properties of individual candidates. Based on the domain analysis in Section 4.1, three types of tags can be distinguished. Initial candidates can be of interest to the architect since they must be observed to determine whether created architectures actual improved or deteriorated. Therefore, *initial* is the first of the tags. The other two tags are related to selection support as described in Section 4.3.3. Thus, the second tag identifies the *Pareto* candidates, while the other highlights candidates *suggested* by software architecture optimization approaches.

4.4.3. Colors & Symbols

To highlight the previously described groups and tags visually, they are mapped to specific colors and symbols. According to Mackinlay [Mac86], mapping data to colors is suitable and precise for nominal data types like these. This allows for easier perception of the group membership and tagged properties.

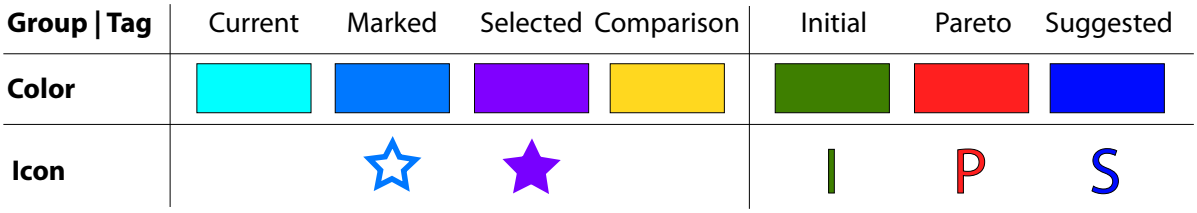

Figure 4.5.: Color and icon mappings for groups and tags in the visualization approach.

The mapping of groups and tags to colors and symbols is shown in Figure 4.5. Different variations of blue are used for the groups *current*, *marked*, and *selected* based on the ordering in a rainbow. As *current* only contains candidates of temporary interest, it is colored in light blue. The interest increases when the *marked* group is used. Thus, the color turns dark blue. The *selected* candidates are of the highest importance to the architect. For this reason, the next color on the rainbow is chosen, which is purple. The yellow color of the comparison group is chosen because it distinguishes from the other

used colors. However, each candidate in this group is mapped to a custom color. Because of this, the color of the group itself is almost irrelevant.

The icon of the *selected* group is a filled star in the group's purple color. The reason for this choice is that the star is an icon that is commonly used to identify favorites. As *marked* candidates are the aspirants for this group, the unfilled star is used as their icon.

The three tags are mostly independent of each other. Therefore, they use the three base colors of the RGB color model for the highest discriminability. The first character in their name is chosen as the symbol. This color and icon mappings assure that it is difficult to confuse them.

4.4.4. Usability

Usability is an important quality attribute of a software system. ISO 25010 [Iso] defines it as:

> "The extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"

Efficiency can, for example, be improved by automation. Concerning the outer interface of the visualization approach, this means that it should be easy for an architect to import or export data. Therefore, architecture optimization approaches should be able to communicate directly with the visualization approach. This relieves the architect from taking care of data formats and data transformations. If candidates are selected for further optimization, the visualization approach can send this information back to the software architecture optimization approach. From the view of the architect, there is almost no perceivable difference between a software architecture optimization approach's built-in visual user interface and the proposed visualization approach, if the communication is automated. Thus, the visualization approach can act as a substitute to the software architecture optimization approach's visual user interface.

The decision to use a custom architectural model, as stated in Section 4.3.2, can have an impact on the architect's satisfaction. This is due to the fact that information can get lost during the transformation to this model, e.g., in contrast to the simplified model, PCM [Reu+11] allows to specify scheduling policies for CPUs. Therefore, being able to export the original model instance is an important feature for the architect. It allows her to investigate the model in full detail and by the use of other approaches.

4.4.5. Global Control

A visualization approach for results of software architecture optimization should not only be a cluster of independent visualizations. To create additional value for the architect, the approach has to assemble the visualizations reasonably. All these visualizations are nothing else than different, visual perspectives on a subset of the same data. Therefore, it is possible to also give the architect an overview of the underlying data. When observing a visualization, the architect should always be able to see and control the most important information about candidates, namely their tags and the groups they belong to. This is important, because grouping is a generic, visualization-independent concept to interact with the candidates. The same applies to global options and filters, e.g., a filter for levels. Any option that is not specific to a single visualization or view has to be accessed and controlled consistently within the whole approach.

4.5. Visualizations

This section presents the mapping of the different data types in software architecture optimization results to visualizations. As elaborated in Section 4.1, these data types can be divided into two categories, namely the goal-based utility values and the architectural description. Appropriate visualization techniques have to be chosen to support the architects in the use cases defined in Section 4.2. Thus, Section 4.5.1 describes visualizations that are suitable for the representation of utility values, while Section 4.5.2 describes the ones suitable for representation of architectural descriptions.

4.5.1. Visualizations for Utility Values

The utility values can be represented as real values ranging from zero to one. Due to the fact that they describe the satisfaction of the architect, they are ordered by definition. Therefore, they can be categorized as ratio-scaled data type. According to Mackinley [Mac86], position and length are appropriate visual properties to represent this kind of data. It needs to be considered that there can be an arbitrary number of specified goals. For each of these goals, a utility value is provided. Therefore, visualizations for multivariate data, as introduced in Section 2.2.2, should be used.

In the following, two kinds of visualizations are distinguished. The first one is more suitable to visualize individual candidates or smaller groups of candidates. The other visualization is more suitable for large groups of candidates and provides an overview.

Thus, they both provide two different perspectives on the same data, which can lead to different insights. For this reason, both types of visualizations should be used.

Candidate-focused Visualizations

For the visualization of high-dimensional, ratio scaled data, multiple types of visualizations could be used, as outlined in Section 2.2.2. Parallel coordinate plots [Mou11] provide some useful properties, e.g., their parallel axes allow for an exact comparison of data by position. However, their drawback is that they grow horizontally when adding more goals. For this work, a more compact representation is desired, because there can be many candidates to display. Therefore, radar charts [DLR09] are chosen as visualization for utility values. Apart from their compactness, they allow for fast recognition of patterns in the data by the shapes of the candidates. Figure 4.6 shows different variations of radar charts used in this work.

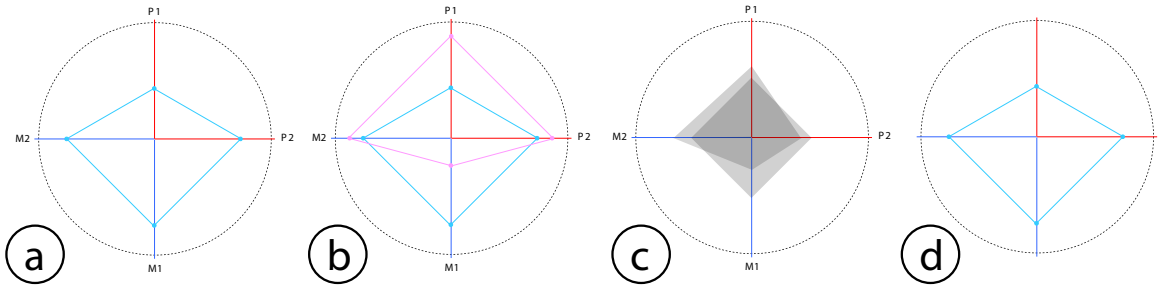

Figure 4.6.: Variations of radar charts showing utility values for (a) single candidates, (b) multiple candidates, (c) populations, and (d) compact representation.

Radar chart (a) is the default variant for the representation of a single candidate. Each axis represents one goal, while axis colors can be used to group goals, e.g., all goals related to performance are colored red. The circles on the axis are located based on the utility values of the candidate. This means that a circle at the outer ring represents a fully satisfied goal, while a circle at the center means that the goal is unsatisfied. These circles are connected by lines. The covered, inner area forms a shape that allows for a quick first impression of the strengths and weaknesses of a candidate. In general, it is also possible to present multiple candidates in one visualization, as shown in radar chart (b). The same color can be used for all candidates to show which values are covered by a group. The comparison group can use the custom color of the candidates to make them distinguishable. Another variation is shown in radar chart (c). In this variation, only the inner areas are painted, but with low opacity. The idea behind this feature is that it is possible to get an impression of how a whole population with many candidates looks like, e.g., a level. Bright areas indicate that only less or no candidates cover this area, while

dark areas are covered by many candidates. Based on this information, the architect can see whether a candidate performs rather better or worse than the whole population. This can be an important piece of information, because a candidate might perform badly with respect to the architect's overall expectations, but still perform better than the rest of the population. In this case, the architect could decide to accept an actually "bad" candidate. Finally, radar chart (d) shows an even more compact variation. If space is rare, potentially long goal names can be removed to save some space. However, such essential information should be available to the architect. If axis names are not shown by default, they must at least be shown when hovering the axes. The same holds for the actual utility values.

Group-focused Visualizations

The previously described radar chart is suitable for a rather small number of candidates. Additionally, the presented solution with reduced opacity of candidates in radar chart (c) is not ideal, because the identification of a specific candidate's values in the chart is nearly impossible. This is acceptable to get a rough overview, but detailed information is missing. A typical visualization for two-dimensional data is the scatter plot [CM84]. It allows to display many value-pairs and the exact values can still be read. For higher-dimensions, a matrix matrix of scatter-plots can be used.

In general, it would be possible to increase the number of displayable dimensions by adding a third dimension or color and symbol mappings. However, the complexity of the scatter plot would also increases quickly. Thus, these techniques would be limited to a small number of goals. The matrix can handle an unlimited number of goals, despite its quadratic grow. In contrast to radar charts and parallel-coordinate plots, the ordering of the axis does not have an important effect. Therefore, it is a useful supplement to the previously described radar chart.

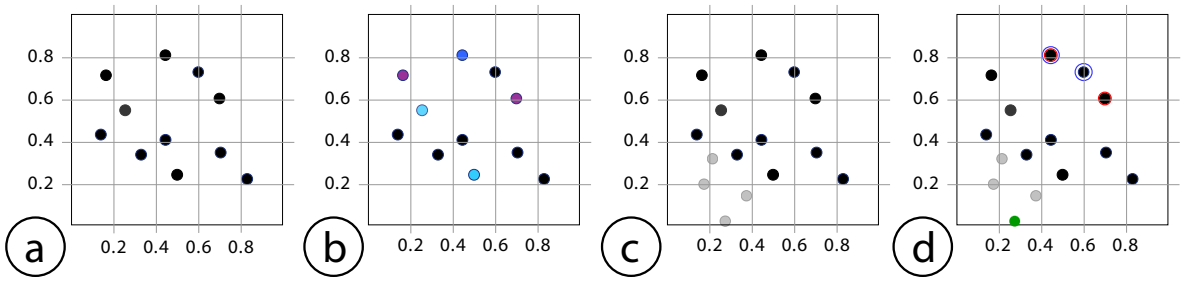

Figure 4.7.: Variations of radar charts showing utility values (a) without color mapping, (b) with color mapping for groups, (c) with parental candidates, and (d) with color mapping for tags.

Four relevant variations of scatter plots are also visualized in Figure 4.7. Scatter plot (a) is the basic versions, in which each circle represents a candidate. The x-axis and the y-axis each represent a single goal. The candidates are then located based on their utility values for these goals. Coloring can be used to show group memberships of candidates, e.g., all *current* candidates are colored in light-blue, as shown in scatter plot (b). Coloring can also be used for the comparison group, but each candidate's circle is then colored in its custom color. Coloring is also a useful method to highlight the same candidate(s) in all scatter plots of the matrix. This makes it easier for the architect to find the same candidate(s) in all scatter plots. Therefore, candidates should be selectable by interaction.

Scatter plot (c) shows another use of coloring. The parent candidates can be shown in the same plot, but with lower opacity. This makes it possible to see the difference between two levels. Finally, scatter plot (d) shows how tags can be visualized. To avoid confusion with the groups' color mappings, suggested and Pareto candidates can be surrounded with rings in their color. This does not hold for the initial candidate, as green or a similar color is not used yet and there is usually only one initial candidate. Thus, the initial candidate is colored in green.

A noteworthy drawback of the scatter plot is that circles can overlap. Circles with low opacity are usually a technique to mitigate this effect, but this technique is already used to display the parents. In addition, this can lead to additive color mixing, which confuses the architect. Therefore, interaction is required to make overlapping circles visible or move a particular circle to the front.

4.5.2. Visualization for Architectures

The architecture is different from the utility values. It consists of three types of information that can be displayed, namely components, allocation, resources. Most of the data can be categorized as nominal scaled, e.g., the components and servers. Only values that describe resources can be ratio scaled, e.g., CPU clock speed. For this reason, they must be treated differently. The according visualizations are presented in the following.

Visualizations for Components

A typical representation for entities and dependencies is a graph. As explained in Section 3.4, it is also common for visualizing software systems. However, it is necessary to display the architectures of many candidates and make comparisons between these architectures. Such comparisons can not efficiently be done by displaying many graphs

next to each other. For this reason, graphs have to be modified by mapping more information to some of their visual properties, e.g., node size.

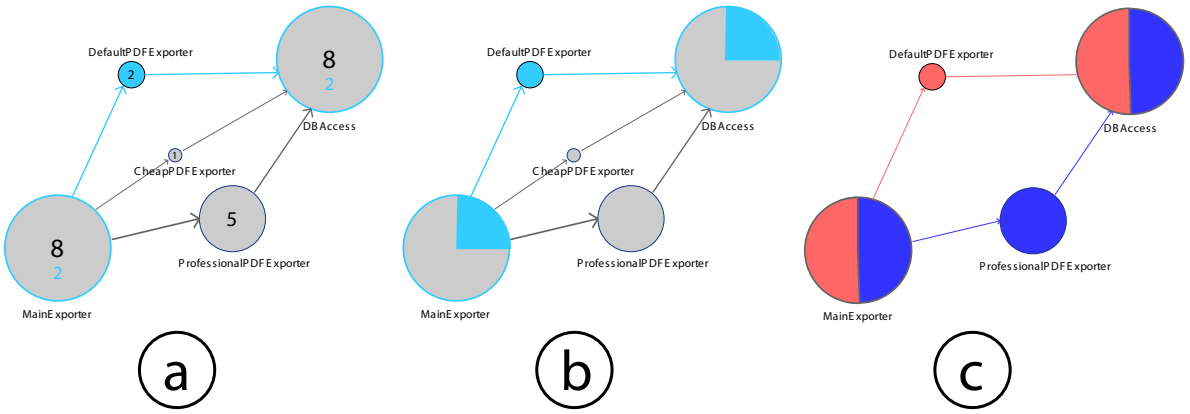

Figure 4.8.: Variations of graphs showing components and dependencies with size mapping and (a) explicit size annotation, (b) implicit size representation by pie charts, and (c) color mapping for *comparison* candidates.

Figure 4.8 shows three variations of graphs relevant to this work. Components are displayed as circles and dependencies between the components are visualized by arrows that connect the components. The radius of a component's circle is chosen based on how many candidates contain the component. To avoid that small circles from being practically invisible and big circles from hiding others, minimum and maximum sizes are defined. The minimum size is applied if a component is used a single time, while the maximum size is applied if all candidates contain the component. Values in between are mapped accordingly in a linear way. This allows the architect to see which components are used more or less often than others. For example, *ProfessionalPDFExporter* is ued by 5 candidates, while the smaller *CheapPDFExporter* is only used by one candidate. As an equation, a circle's radius is computed in the following way:

$$radius_{component} = radius_{min} + \frac{\text{Number of using candidates} - 1}{\text{Number of all candidates} - 1} \cdot (radius_{max} - radius_{min})$$

The same mapping is also applied to the thickness of the arrows based on the presence of dependencies in the candidates' architectures. Textual annotations show the exact numbers of candidates that contain an element and the names of components. All the associated candidates can be highlighted when clicking at a circle or arrow. For example, in Graph (a), *DefaultPDFExporter* and its two candidates are highlighted. Both candidates also contain the components *MainExporter* and *DBAccess*. This is indicated by the colored annotations.

Graph (b) displays the same information by the use of pie charts instead of textual annotations. Pie charts are a suitable and compact way when showing part-of-a-whole relationships. Despite their rather low accuracy, they still allow the architect to see abnormal values. They also perfectly fit in the circles. Of course, the exact number of candidates has to be shown when hovering the circles and arrows. Graph (c) shows how the graph can be used to present the *comparison* group. An entity is only contained in a candidate's architecture if the candidate's color is present. For example, in Graph (c), the red candidate contains the *DefaultPDFExporter*, while the blue one contains the *ProfessionalPDFExporter*.

Architectures with many components and dependencies likely suffer from visual cluttering. Therefore, a strategy to handle visual cluttering has to be implemented to support architectures of big software systems. A common strategy is to use a force-directed layout [FR91], and allow the architect to edit the graph, e.g., by filtering components or modifying the graph's layout.

Visualizations for Allocation

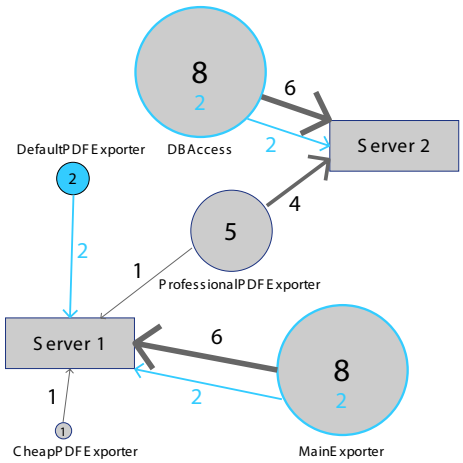

Figure 4.9.: Example of a graph showing the allocation of components to servers.

As for components, allocation can be visualized by a graph. Therefore, mapping of sizes, interaction, and highlighting can be applied as described in the previous section. An example is shown in Figure 4.9. Circles are again the components, rectangles are servers, and lines represent the allocation. In the example, *ProfessionalPDFExporter* is allocated to *Server 2* by four candidates, while one candidate allocates it to *Server 1*.

Visualizations for Resources

In the proposed approach, resources can be described by real numbers, e.g., CPU clock speed or latency in a network. In contrast to the utility values, these values do not have a common range or optimization direction. For example, while clock speed is usually maximized, latency is minimized. Therefore, a radar chart is not a suitable choice, as there is not much space for placing individual range captions. The parallel coordinate plot provides more space and is also suitable for showing many candidates in a single plot.

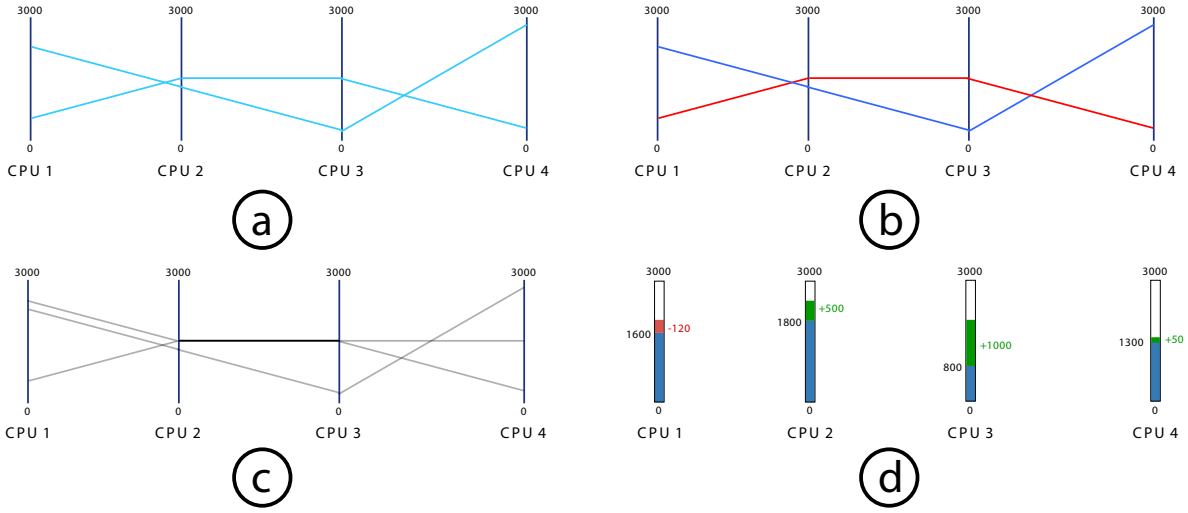

Figure 4.10.: Variations of parallel coordinate plots and a bar chart showing resource values for (a) groups, (b) comparison candidates, (c) populations, (d) single candidate with changes.

Figure 4.10 shows three variations of the parallel coordinates plot and one alternative using bar charts. The plots each visualize the clock speed of four CPUs, represented by four axes. Plot (a) uses the same color to display two candidates of the *current* group. One candidate is drawn as a line which crosses each axis at a point that is based on the candidate's CPU clock speed. Plot (b) shows a method to visualize two members of the *comparison* group based on their custom colors. Here, one candidate is colored in blue, while the other is colored in red. Plot (c) shows a typical problem in parallel coordinate plots: lines are covered by other lines. To mitigate this case, lines can be drawn with low opacity, so they appear darker at locations of an intersection. Another problem is that the path of a line can appear ambiguous due to intersections. Therefore, a hovered line should be highlighted, e.g., by making it thicker. Plot (d) presents an alternative using bar charts when a single candidate is compared to another candidate. Differences

between the resource values can be visualized by extra bars in a different color, e.g., red can be used to show a decrease, while green shows an increase.

4.6. Views

Visualizations for different kinds of data have been proposed in Section 4.5. These visualizations can be used as the building blocks to create views, which can support the architect in the use cases defined in Section 4.2. By assembling visualizations into views, these views can take advantage of the strengths of various visualizations. Therefore, each view can be designed to provide a different perspective on the data, which makes it valuable for solving specific use cases.

For this work, several drafts for views have been created. Section 4.6.1 describes elements to control the groups and global options for all other views. Section 4.6.2 and Section 4.6.3 present the Population View and the Candidates View, which both provide different perspectives on utility values. The Architecture View, which is introduced in Section 4.6.4, focuses on the visualization of architectural data. Then, Section 4.6.5 presents the Candidate View, which can be used to investigate the evolution of a single candidate. A view to manage goals, namely the Goals View, is introduced in Section 4.6.6. Finally, Section 4.6.7 outlines the need for administrative views to manage projects.

4.6.1. Control Views

The need for global control has already been stated in Section 4.4.5. The toolbar is shown in all views inside a project. It assures that there is always a consistent way to control the data and to access the project settings. A possible design is illustrated in Figure 4.11.

The toolbar allows to assign candidates to different groups, as shown in Figure 4.11 (a). The visible groups are the same as introduced in Section 4.4.1, using the same colors and symbols as introduced in Section 4.4.3. Each group has its own label in the toolbar and can be collapsed. The number of candidates inside a group is shown behind its name. *All* contains all currently visible candidates. In the example, it consists of 3456 candidates.

Each group contains a list of its group members. Candidates are identified by a unique number, which is part of their visible name. Candidates can be assigned to the *marked* or *selected* group by clicking at the star in front of their name. In the *comparison* group, this element is replaced with a selection dialog for the candidate's custom color. The tags

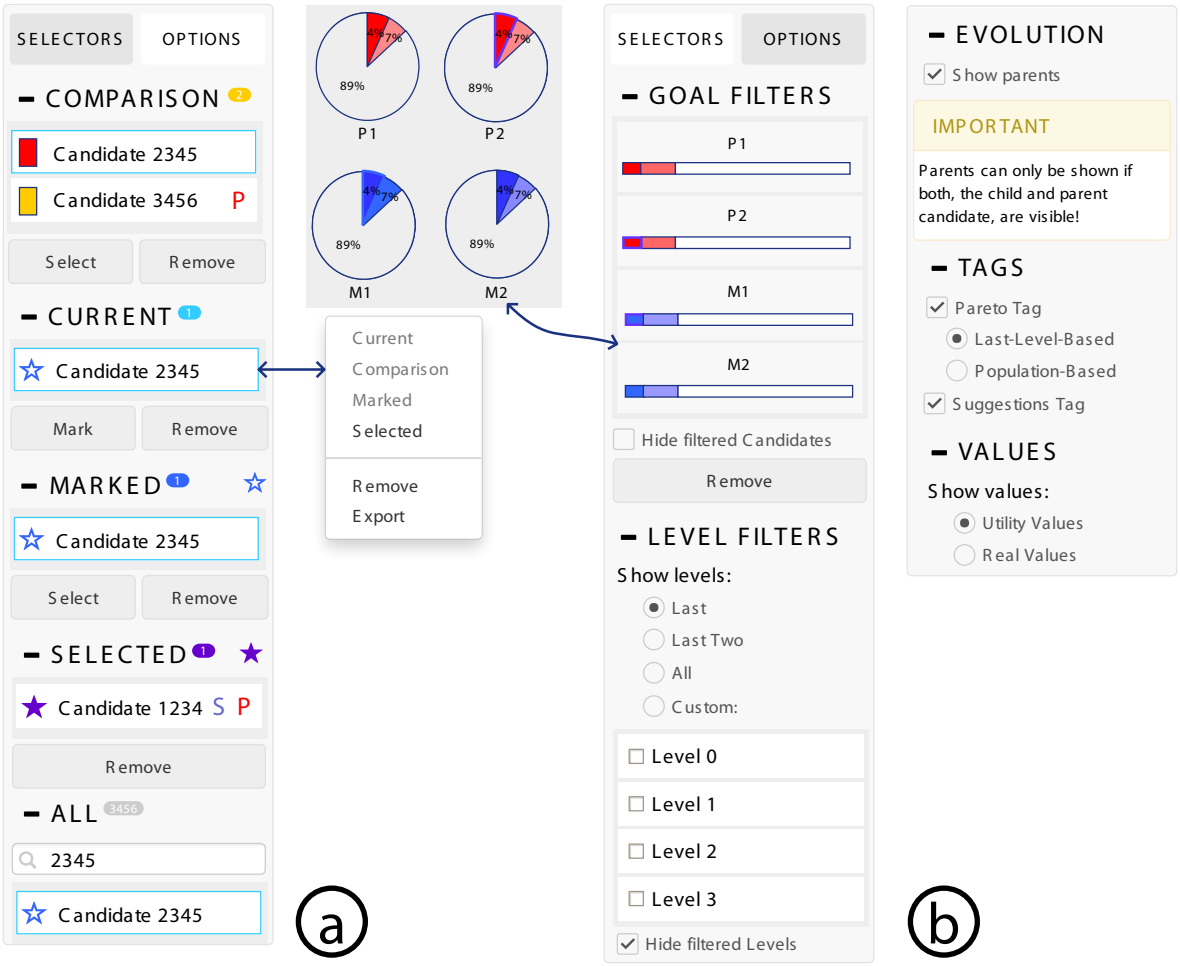

Figure 4.11.: Design of the toolbar consisting of tabs for (a) group control and (b) managing global settings and filters.

described in Section 4.4.2 are visualized as colored icons behind their candidate's names. For example, *Candidate 1234* is suggested by the software architecture optimization approach and also a Pareto candidate, according to its tags.

The *all* group allows searching for candidates by their identifier or their tag. Buttons at the bottom of a group's list can be used to control the whole group, e.g., in the *current* group, all contained candidates can be grouped as *marked* or removed from the *current* group. In addition, a context menu can be opened by right-clicking at a candidate. This allows for the control of individual candidates. They can be grouped, ungrouped, or exported.

Examples for project settings are shown in Figure 4.11 (b). Pie charts or bar charts can be used to present the percentage of fulfilled and almost fulfilled goals. The architect can click on areas to apply global filters, e.g., only candidates that fulfill goal *P2* and goal *M1* should be visible. Another option is the level filter. By default, only the last level is visible, but the architect can also specify which levels should be visualized. The architect can decide whether parents or tags should be shown. A Pareto analysis can also be applied to each level individually or the whole population. A future version might also allow switching between the actual response values and utility values, while this is not supported in the initial version of the approach, as only utility values are visualized. In addition to the shown toolbar, an actual implementation also needs a dialog for sending back candidates from the visualization approach to the software architecture optimization approach.

4.6.2. Population View

The Population View has been designed to rather provide an overview of all candidates and groups of candidates. It can also be used to make patterns and dependencies between goals visible. Therefore, it relies on visualization and interaction techniques discussed in Section 4.5.1. Figure 4.12 illustrates the complete view. The toolbar described in the previous section can be attached at the right or left side of the view.

On the right side of the view, a matrix contains the plots for all combinations of two goals. In the example, there are three goals present, namely *P1*, *M1*, and *R1*. Plots in a horizontal line have the same goal on the y-axis, and plots in a vertical line have the same goal on the x-axis. For example, the highlighted plot in the top left corner maps *R1* to the x-axis and *P1* to the y-axis.

A magnified version of one of the plots can be shown on the left. The architect can select which plot is shown here by clicking at one of the scatter plots in the matrix. Colors are used to indicate group membership and tagging as described in Section 4.5.1. Clicking at candidates groups them as *current* and adds arrows to the magnified scatter plot. These arrows connect *current* candidates and their parents to visualize the improvement of these candidates. A radar chart, as described in Section 4.5.1, is shown for each *current* candidate at the bottom of the view. In the matrix, architects would have to search for a single candidate in each plot. Therefore, the radar charts should help to see the strengths and weaknesses of each candidate in a much more compact representation without switching to another view.

Highlighting can be adjusted by the options in the top left corner of the view. Here, the architect can select which color mappings should be applied, e.g., she could decide to

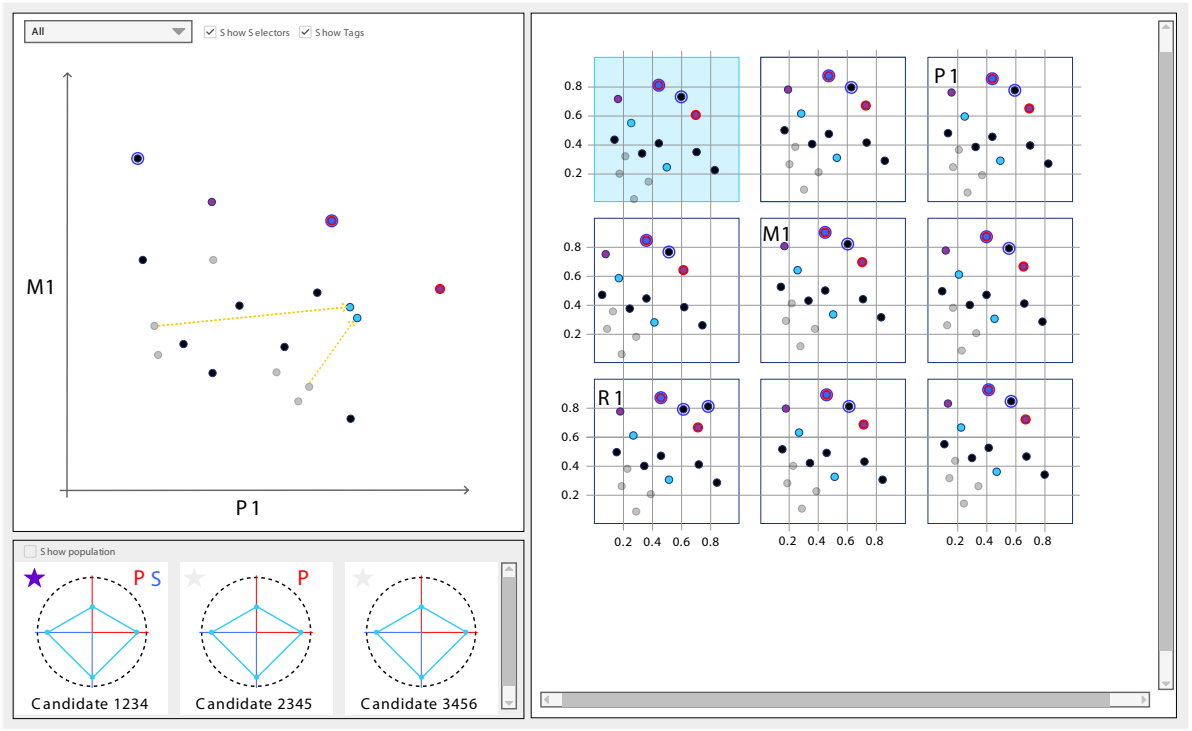

Figure 4.12.: Design of the Population View. Right: Matrix of scatter plots for all combinations of goals. Left: Magnified scatter plot and radar charts for *current* candidates.

only highlight members of the *selected* group. Besides, architects can deactivate tags or the arrows that show parental relationships.

The properties of this view are summarized in Table 4.2. This view only visualizes utility values and no architectural information. Due to the matrix, it is easy to get an overview, but detailed comparisons between a small number of candidates are hard. However, the use of radar charts mitigates this drawback to some extent. Interrelationships between two goals can be quickly detected if patterns in the scatter plots appear, e.g., circles form clusters. Besides, it can be seen whether circles are located above a utility threshold. Thus, the architect can investigate satisfiability of goals. The view also visualizes evolutionary data, as parents can be shown as circles, and changes of the utility values are displayed as arrows. The architect can then see whether a candidate improved. However, arrows are only visible in the magnified scatter plot and they can be covered by other circles, which makes it harder for architects to find the desired, evolutionary information for a specific candidate.

With regard to the use cases defined in Section 4.2, this view can support the architect in all use cases, except for the implementation of architects (*UC3*), as no architectural

Category	Subcategory	Population View
Data Type	value	● ● ●
	architecture	○ ○ ○
Comparison	detailed	● ○ ○
	overview	● ● ●
Interrelationship	candidate/goal	● ● ○
	candidate/architecture	○ ○ ○
Evolution	parents	● ● ○
	changes	● ○ ○

Table 4.2.: Usefulness of the Population View for tasks with regard to visual properties. The usefulness of each property is expressed on a 0-3 scale, where 0 is *not useful* and 3 *very useful*. ○ has a value of 0, ● has a value of 1 and can be summed up.

information is displayed. For selecting candidates (*UC1*), the overview can be useful to reduce the number of potential candidates. While evolutionary information can help architects to decide whether to continue the search (*UC2*), interrelationships can help to explain why results do (not) fulfill the specified goals (*UC4*).

4.6.3. Candidates View

The Candidates View uses different variations of the radar charts introduced in Section 4.5.1. Therefore, it focuses on a more detailed analysis of individual candidates. The view is illustrated in Figure 4.13.

The big radar chart on the left shows the name of the goals at the end of the axis. To save space, the smaller radar charts on the right require interaction to see the axis name. In addition, hovering the circles on the axis shows the actual value. In the list on the right, each candidate has its card. The behavior is quite similar to the lists in the toolbar in Section 4.6.1. It uses the same tags, the same right-click menu, and candidates can be grouped by clicking at the star in the top left corner. Also, the search is the same and enables the search for identifiers and tags. In addition, the dropdown menu on the top right corner can be used to filter the list. Clicking at a candidate's card groups this candidate as *current*.

The big radar chart on the left also displays candidates. The difference is that it shows a whole group and not individual candidates. This representation is, therefore, better suited for comparison of candidates. The group to display can be chosen in the dropdown

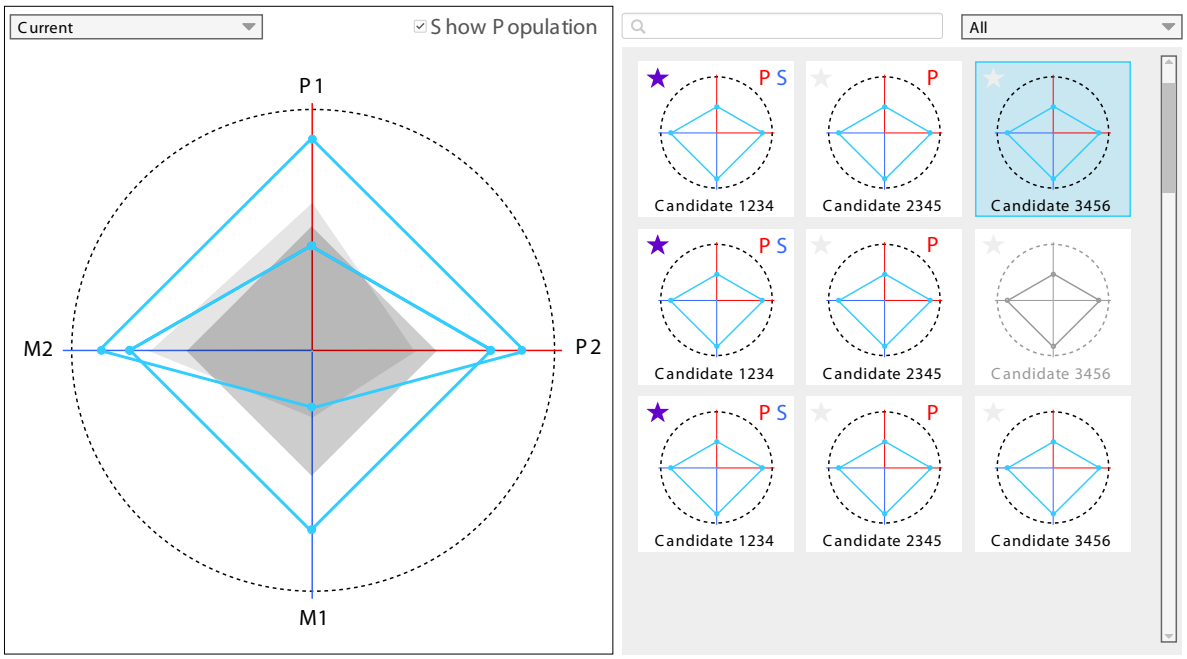

Figure 4.13.: Design of the Candidates View. Right: Radar charts of individual candidates. Left: Radar chart showing groups of candidates and the population.

menu above, e.g., the *current* group is active in the example. Switching to the *comparison* group changes the colors of the candidates, which makes comparisons even easier.

In the background, all candidates of the visible levels are shown, but only with low opacity. As explained in Section 4.5.1, this feature should allow the architect to get an impression of how the whole population looks like. Thus, a group or single candidate can be compared to the population.

Table 4.3 summarizes the properties of the Candidates View. As the Population View, this view only displays utility values and no architectural information. As explained before, this view is rather designed for detailed comparisons of small numbers of candidates and detailed investigation of single candidate's utility values. However, the transparent candidates that are shown in the background also provide some information about the population, although specific candidates can not be identified in the representation anymore. Still, it is possible to see whether the goals are satisfied for a level.

With regard to the use cases defined in Section 4.2, this view is intended to assist the architect in selecting candidates (*UC1*). Information about the whole level can also be useful in finding a stopping criterion (*UC2*), because improvements and degradation becomes visible. As it does not contain architectural information, this view is not suited to show how candidates can be implemented (*UC3*) and explained (*UC4*).

Category	Subcategory	Candidates View
Data Type	value	● ● ●
	architecture	○ ○ ○
Comparison	detailed	● ● ●
	overview	● ○ ○
Interrelationship	candidate/goal	● ○ ○
	candidate/architecture	○ ○ ○
Evolution	parents	○ ○ ○
	changes	○ ○ ○

Table 4.3.: Usefulness of the Candidates View for tasks with regard to visual properties. The usefulness of each property is expressed on a 0-3 scale, where 0 is *not useful* and 3 *very useful*. ○ has a value of 0, ● has a value of 1 and can be summed up.

4.6.4. Architecture View

While the previous two views focused only on utility values, this view is designed to explore the architecture of the candidates. Therefore, it combines the visualizations and behaviors introduced in Section 4.5.2. The view is illustrated in Figure 4.14.

The visualization showing the system's components and their dependencies is placed on the bottom on the left side. To its right, the visualization that shows component allocation is shown. A force-directed layout is used to place the elements and architects can drag components. Right-Clicking a component or dependency, will group all its candidates as *current*. Then, a component's edge is highlighted in light-blue if the component is present in at least one of the *current* candidates as explained in Section 4.5.2. Highlighting is also applied to the visualization for resources on the top. Finally, the radar chart in the top left corner shows all *current* candidates. It allows seeing the utility values while exploring the architecture.

In order to change the active candidates for this view, the dropdown menu in the top left corner can be used, e.g., the group can be changed from *all* to *selected*. In this case, only the *selected* candidates' architectural information is considered by the visualizations. Furthermore, the *comparison* candidates can be selected, which changes the appearance of the view as shown in Figure 4.15. The colors allow the candidate to see which candidate contains which element.

For the graphs, the *reduce graph* feature can be activated. Then, only the differences between the architectures remain. This feature can also be used outside of the comparison mode. However, common elements for a bigger amount of candidates are usually rare, so

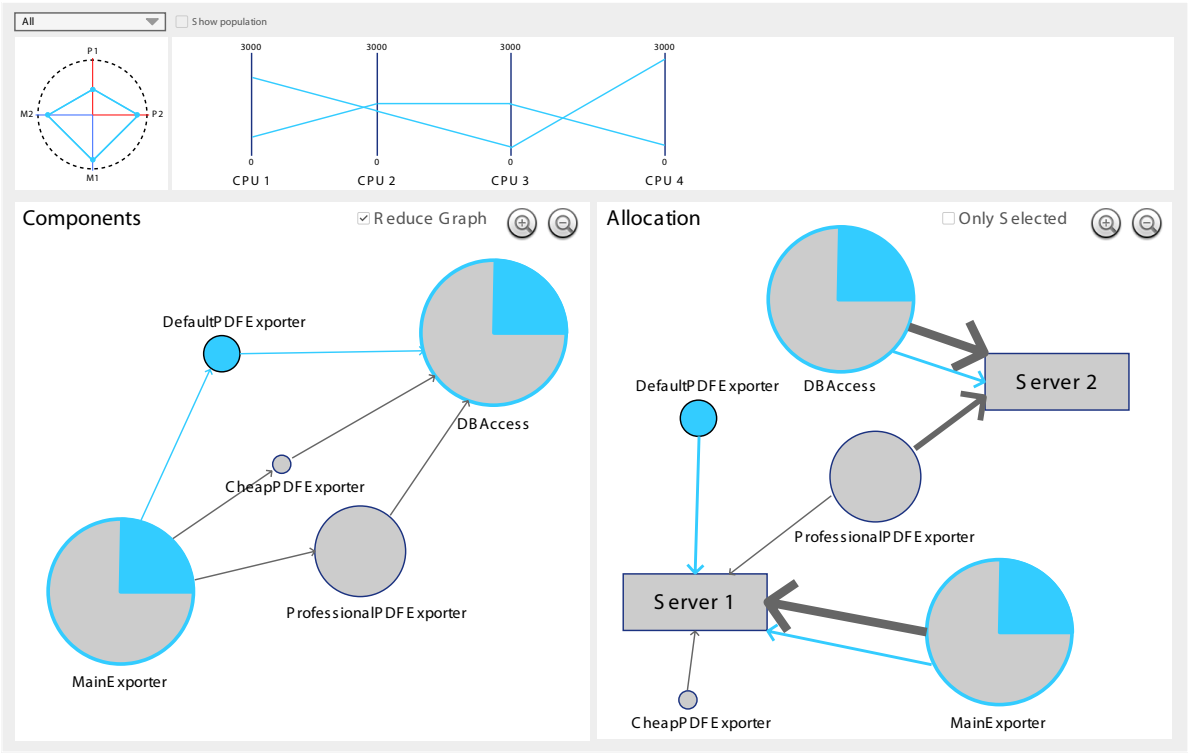

Figure 4.14.: Design of the Architecture View. Bottom: Graphs showing components and dependencies, as well as allocation. Top: Radar chart showing utility values of *current* candidates and parallel coordinate plot showing resource values.

there might be no visible effect. In this case, there is also the option to apply additional filtering when the reduce graph feature is active. The smallest and biggest candidates can be filtered and link length and strength can be set to modify the appearance of the graph.

The properties of the Architecture View are concluded in Table 4.4. This view shows all available architectural information, while utility values are just presented in a single, small radar chart. On the one hand, the comparison mode and the *graph reduction* feature can be used to compare small numbers of candidates in detail. On the other hand, the view can show all candidates of large groups compactly by mapping numbers to the sizes of graph elements. As utility values and architectural information are both visible in the same view, it is possible to see the interrelationship between these data types.

With regard to the use cases defined in Section 4.2, this view can be used to determine the necessary changes to implement a candidate (*UC3*). In addition, it can be used to explain the results (*UC4*) by investigation of the interrelationship between utility values

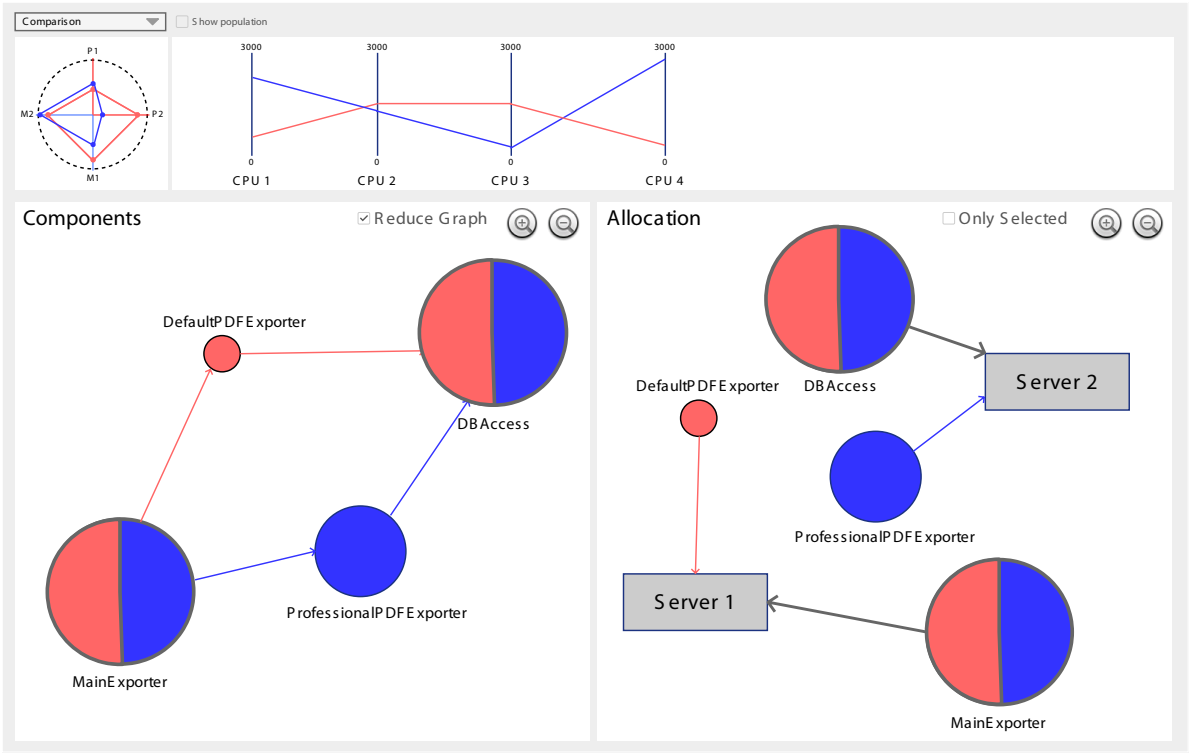

Figure 4.15.: Design of the Architecture View in comparison mode for two candidates. Bottom: Graphs showing components and dependencies, as well as allocation. Top: Radar chart showing utility values of *comparison* candidates and parallel coordinate plot showing resource values.

Category	Subcategory	Architecture View
Data Type	value	● ○ ○
	architecture	● ● ●
Comparison	detailed	● ● ●
	overview	● ● ●
Interrelationship	candidate/goal	○ ○ ○
	candidate/architecture	● ● ○
Evolution	parents	○ ○ ○
	changes	● ○ ○

Table 4.4.: Usefulness of the Architecture View for tasks with regard to visual properties. The usefulness of each property is expressed on a 0-3 scale, where 0 is *not useful* and 3 *very useful*. ○ has a value of 0, ● has a value of 1 and can be summed up.

and architectures. If the architect wants to consider architectural information, it can also be relevant when selecting candidates (*UC1*) or searching for a stopping criteriion (*UC2*).

4.6.5. Candidate View

The Candidate View is a variation of the previously introduced Architecture View from Section 4.6.4. It only visualizes a single candidate and adds support for evolutionary information. The view is illustrated in Figure 4.16.

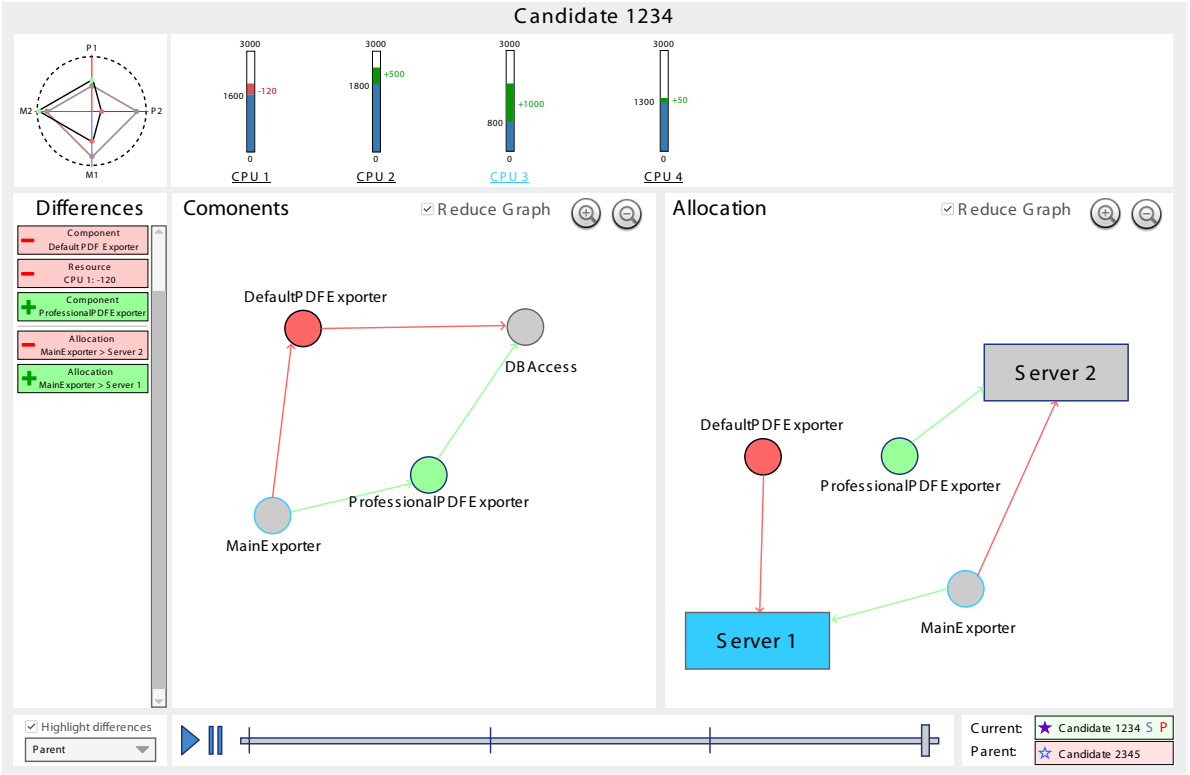

Figure 4.16.: Design of the Candidate View. Bottom: Timeline to select parental candidates. Middle: List of differences and graphs showing the changes for components and dependencies, as well as allocation. Top: Radar chart showing utility values of the active candidate and its parent, as well as a bar chart showing changes in resource values.

A major difference is the timeline at the bottom. It can be used to select the parental candidates. Thus, this view allows for scrolling through the evolution of a candidate and the applied changes. it also shows which changes have been applied to the parent

Category	Subcategory	Candidate View
Data Type	value	● ○ ○
	architecture	● ● ●
Comparison	detailed	● ● ●
	overview	○ ○ ○
Interrelationship	candidate/goal	○ ○ ○
	candidate/architecture	● ● ○
Evolution	parents	● ● ●
	changes	● ● ●

Table 4.5.: Usefulness of the Candidate View for tasks with regard to visual properties. The usefulness of each property is expressed on a 0-3 scale, where 0 is *not useful* and 3 *very useful*. ○ has a value of 0, ● has a value of 1 and can be summed up.

candidate to create the child candidate. In general, green color is used to highlight added elements, while red color highlights removed elements.

The graphs for components and allocations basically stay the same. The only difference is that the size mapping becomes obsolete and the colors now show the change between parent and child candidates. The parallel coordinate plot has been replaced with bar charts, as only one candidate and its changes are shown. The radar chart also uses the described color mapping. For improved utility values, the circle is colored green, while it becomes red for deteriorated values. The lines of the parent candidate are colored gray, to allow for better discriminability of the candidates. A list of differences is shown on the left. Here, differences detected by the analysis of the visualization approach can be shown. A future version of the approach could also allow software architecture optimization approaches to submit their logged changes for more detail.

Table 4.5 summarizes the properties of the Candidate View, which are similar to the Architecture View. Of course, due to the reduction to a single candidate, the overview gets lost. However, evolutionary information is added, because parents can be selected by clicking at the timeline. In addition, changes are highlighted through the modified color mapping.

With regard to the use cases defined in Section 4.2, this view satisfies the needs of architects by investigating the necessary changes to implement a specific candidate (*UC3*). It can also be used to trace the evolution of candidates, and therefore, see which changes led to its improvement or degradation for particular utility values. However, it lacks the overview for finding a stopping criterion (*UC2*) and the focus on utility values that is required to select candidates (*UC1*).

4.6.6. Goal View

The Goal View is designed to manage goals and to visualize statistical data related to these goals. The option to deactivate goals can be useful to reduce the amount of visual information in visualizations based on utility values. For example, this can be useful if a goal is always fulfilled or the architect wants to find trade-offs based on other goals first. Figure 4.17 illustrates the Goal View.

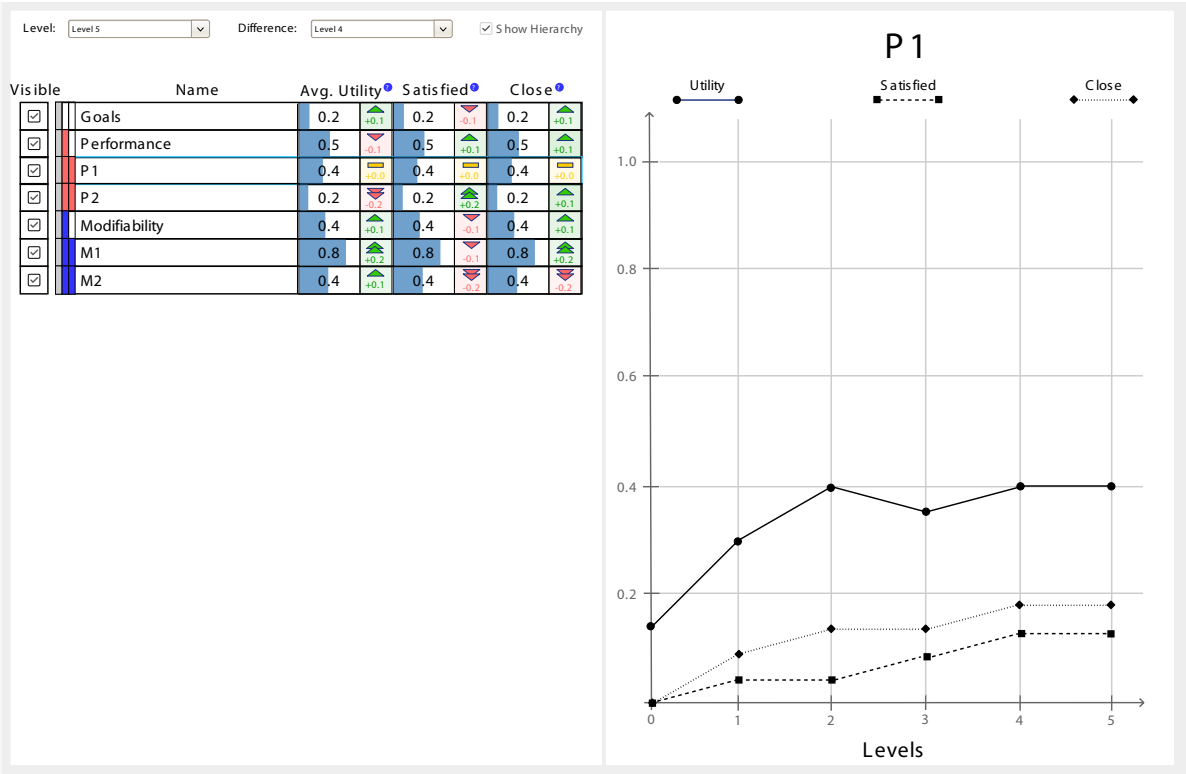

Figure 4.17.: Design of the Goal View. Left: Metrics for individual goals and differences between two levels. Right: Metrics for single goal over all levels.

This view basically uses three metrics that are computed for whole levels. The first one is the average utility value for each goal. The other two are the same as used by the *Goal Filters* shown as part of the toolbar in Section 4.6.2. The percentage of candidates that fulfill and almost fulfill a goal is computed. To determine whether a goal is almost fulfilled, a threshold, e.g., a utility value of 0.8, can be defined.

On the left, these values are displayed for each of the goals. The blue bar charts visualize these values and allow for a quick comparison of the goals' metrics. Icons behind the values show the change compared to the previous level. Green triangles pointing upwards indicate improvement, while red triangles pointing downwards illustrate degradation.

Category	Subcategory	Goal View
Data Type	value	● ○ ○
	architecture	○ ○ ○
Comparison	detailed	○ ○ ○
	overview	● ● ●
Interrelationship	candidate/goal	● ● ●
	candidate/architecture	○ ○ ○
Evolution	parents	● ○ ○
	changes	● ○ ○

Table 4.6.: Usefulness of the Goal View for tasks with regard to visual properties. The usefulness of each property is expressed on a 0-3 scale, where 0 is *not useful* and 3 *very useful*. ○ has a value of 0, ● has a value of 1 and can be summed up.

Two triangles indicate the biggest change for a metric. A colored icicle plot [KL83] shows the hierarchy of the goals, as well as their assigned color. The color can be changed by clicking at the colored boxes in the icicle plots. In addition, checkboxes allow for filtering goals in the views. On the right, the metrics for a particular goal can be plotted against the levels. Thus, the evolution of goal satisfaction can be perceived by the architect.

The properties of the Goal View are summarized in Table 4.6. The shown metrics are based on utility values, but rather in an aggregated form. Therefore, this view is suited to provide an overview but does not provide details on the level of individual candidates. The same holds for the evolutionary information, which is available for whole levels, but not for individual candidates.

With regard to the use cases defined in Section 4.2, this view mainly fulfills the requirements for deciding whether to stop the optimization (*UC2*). In this use case, the architect can use the metrics to justify her decisions, e.g., stop if no (significant) changes take place any more. The view can also help to select candidates (*UC1*) by reduction of active goals. This can make it easier to find suitable candidates. However, the view does not provide noteworthy support for the detection of changes for the implementation (*UC3*) and explaining the results (*UC4*).

4.6.7. Administrative Views

The previously described views are designed to visualize data, but administrative views are also required for certain tasks, e.g., selecting a project. However, they are not as essential as the visualization views and their design should fit the used technologies. For

this reason, no generic designs are proposed. Nevertheless, these views have to be part of an actual implementation.

One view is required to select a project for further investigation by using the visualization views. This can be a simple list showing the projects' names, but also present a preview on the projects' metadata, e.g., the number of loaded levels and candidates. In addition, a view to observe a project's settings and metadata can be a useful addition. It is also necessary to extend the visualization views with techniques to navigate between the views.

Chapter 5

Prototype

A prototype of the concept for a visualization approach presented in Chapter 4 has been developed to asses the concept in practice. Although the proposed visualization approach is not bound to a specific software architecture optimization approach, the prototype focuses on SQuAT [Rag+17] as a reference implementation. This also means that the prototype only supports the import of SQuAT's [Rag+17] underlying architectural model, namely PCM [BKR07]. Due to focus on the SQuAT [Rag+17] approach, the prototype is named *SquatVis*. However, the prototype is not strictly bound to SQuAT [Rag+17]. Due to its generic interface, data from other approaches can be imported as well. However, some features might not be available, e.g., architectures can not be analyzed and visualized if they are not provided as an instance of PCM [BKR07]. The implementation of this prototype is publicly available [Fraa; Frab].

The decision to use SQuAT [Rag+17] as the reference implementation of software architecture optimization approaches has multiple reasons. The most important one is that SQuAT [Rag+17] supports all of the proposed features described in Chapter 4, e.g., it logs parental candidates and makes suggestions. Additionally, it can be used for interactive optimization due to the generation of candidates in levels. For this reason, all the four use cases described in Section 4.2 are relevant in the context of SQuAT [Rag+17]. Vice versa, SQuAT [Rag+17] would also benefit from a user interface as it does not have one, yet. However, visual representation is a prerequisite for efficient interactive optimization. Thus, a visual user interface for SQuAT [Rag+17] would also push the research on interactive optimization in the domain of software architecture optimization forward.

In the following, Section 5.1 outlines the specific requirements for the development of the prototype. Section 5.2 describes which technologies have been used to meet the previously described requirements. Then, Section 5.3 presents the top-level design of *SquatVis*. The remaining sections focus on different aspects of this system in more detail. Firstly, Section 5.4 describes parts of the underlying data structure of the

system. Secondly, Section 5.5 focuses on the outer interface and the communication with software architecture optimization approaches. Finally, Section 5.6 presents the implemented views and points out differences to the conceptual design.

5.1. Requirements

The functional requirements for the development of *SquatVis* mostly arise from the conceptual description in Chapter 4. The initial version of *SquatVis* even reduces the functionality, because the Candidate View (see Section 4.6.5) and Goal View (see Section 4.6.6) are not part of the implementation. This decision has been made to reduce the effort that is necessary and get a running prototypical implementation more quickly. Thus, architects can also provide feedback earlier and help to improve the presented approach. While this decision might impact the architects' performance in finding a stopping criterion (see Section 4.2.3) and changes for the implementation (see Section 4.2.4), the other use cases should not be affected. Still, it is expected that all use cases can be handled by the architect, even with the reduced set of views.

For a prototype, not only the functional requirements are important. Suitable technologies have to be selected and the system's architecture has to be defined. To justify the made decisions, also non-functional requirements have been considered. These requirements are described in Section 5.1.1. The functional and non-functional requirements also lead to rather technical requirements, which already have a strong influence on the actual implementation. These technical requirements are described in Section 5.1.2.

5.1.1. Non-Functional Requirements

Four software quality attributes with high importance for this work have to be emphasized. The prototype needs to achieve high **interoperability**, **modularity**, **portability**, and **modifiability**. The rationale for selecting these quality attributes is explained in the following:

According to ISO 25010 [Iso], **interoperability** is defined as

> "degree to which two or more systems, products, or components can exchange information and use the information that has been exchanged."

Interoperability is important for this work, because the visualization approach needs to exchange information with various software architecture optimization approaches. It must receive the results provided by these approaches and be able to send candidates

for the next level. For reasons stated in Section 4.4.4, this should happen with minimal inclusion of the architect.

According to ISO 25010 [Iso], **portability** is defined as

> "degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another."

In general, it can not be guaranteed that software architecture optimization approaches all run in the same environment, e.g., they might run only on some operating systems or inside of virtual containers. To ensure that the prototype can also be used in the same environments, it should not be bound to a particular environment.

According to ISO 25010 [Iso], **modularity** is defined as

> "degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components."

Modularity in this project can be applied in two contexts. In this definition, *SquatVis* and the software architecture optimization approaches can be seen as components. As this work aims to provide a general concept for visualizations in the context of software architecture optimization. Changes to one of these "components" should reasonably have minimal impact on the others. For the conceptual integration in distributed implementations of software architecture optimization approaches, like the containerized implementation of SQuAT [PRF17], high modularization is even more important. In the ideal case, *SquatVis* should be able to run in its own container. However, the definition of modularity can also be applied to the inner structure of *SquatVis*. Here, it is also important, because the approach must be extendable to react to the architects' demands. A modular architecture limits the changes to only some parts of the system and facilitates the implementation of improvements.

According to ISO 25010 [Iso], **modifiability** is defined as

> "degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality."

As explained before, the prototype needs to be extendable, e.g., views and visualizations might have to be modified or added. Besides, support for other architectural models is planned. Therefore, modifiability is important in the process of improving the visualization approach.

5.1.2. Technical Requirements

Based on the functional and non-functional requirements, technical requirements can be derived. They already describe the necessary parts of the system on an abstract level. Five system parts are outlined in the following.

- **Interface:** To achieve modularity, an interface must be defined for *SquatVis*. It has to define the provided operations and data types, which can then be used by the server- and the client-side. Thus, it is reasonable to provide this interface as an own library, which has to be imported by *SquatVis* and the connected software architecture optimization approaches.
- **Communication:** To achieve interoperability, communication should mostly be abstracted. This means that the interface library has to provide client-side operations that can be used by software architecture optimization approaches. Thus, the developer only needs to transform the data to fit the common data types and hands it over to the provided methods. Then, the communication on the technical level is already part of the library, so the developer does not have to take care of it.
- **Visualizations:** Visualizations are an essential part of a visualization tool. Therefore, the choice of a visualization framework is a vital decision. On the one hand, it needs to support the designed visualizations. On the other hand, it should be flexible and allow to quickly build modular visualizations to facilitate future improvements.
- **Interaction:** Some functionalities of the approach require the support of interaction with global effect, e.g., a level filter influences the shown data in all visualizations. Thus, a technology for interaction is required. Besides, visualizations must be able to communicate with the rest of the application.
- **Database:** Software architecture optimization approaches might run for many hours, or even days. For this reason, the results should be stored until the architect actively decides to delete them. From the architect's perspective, it is not acceptable to lose these gathered results too early, e.g., due to an application restart. Therefore, it is necessary to store these results in a database.

5.2. Technologies

As explained before, the implementation of *SquatVis* relies on the prototypical implementation of SQuAT [Rag+17] as a reference implementation of a software architecture

optimization approach. The underlying architectural model is PCM [BKR07], which must be supported by *SquatVis*. Due to the fact that SQuAT [Rag+17] and PCM [BKR07] are both implemented in Java [Orab], this decisions is adopted for *SquatVis*. Thus, technological incompatibility is avoided by design. Furthermore, Java [Orab] is widely used, platform-independent, and a great number of libraries are available.

For the front-end, the JavaScript visualization library D3.js [BOH11; Bos] has been chosen for the generation of visualizations. D3.js [BOH11; Bos] stands for "Data-Driven Documents" and is a state-of-the-art, open-source library for visualization in the web. A big user community provided lots of example visualizations and also the proposed visualizations have been implemented with D3.js [BOH11; Bos] before. Thus, it is possible to build on existing work. Furthermore, extensions are available and interaction can be added with JavaScript.

Due to the use of D3.js [BOH11; Bos] as visualization library, the front-end has to be based on web technologies [Duc14]. Thus, the views are built in HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript to be shown in a browser. Additionally, the responsive front-end component library Bootstrap [TI] is used to design the views. The use of web technologies also contributes to the prototype's modularity, as the creation of visualizations and views mostly takes place in the architect's browser, while the back-end provides the data and the business logic. Therefore, front-end and back-end could even be run on independent machines.

The back-end of *SquatVis* relies on JSF [Oraa] (Java Server Faces). It is the built-in standard of the Java Enterprise Edition [Orab] for the development of Java-based web applications. It generates the front-end web pages from templates, so-called Facelets, and allows to specify business logic in modular units, so-called beans. Thus, this setup naturally contributes to the approach's modularity and modifiability. For the communication with the generated web pages Ajax [Gar+05] (Asynchronous JavaScript and XML) is used. Hence, the back-end also supports interaction with the front-end. Several components are also provided by the JSF framework PrimeFaces [Pri].

For the communication with the software architecture optimization approaches, Java [Orab] provides so-called sockets for the communication between two computers based on TCP [Pos81] (Transmission Control Protocol). Moreover, the JPA (Java Peristence API) provides the support for databases with object-relation mapping. In this work, the database ObjectDB [Sof] is used, because it is comparatively fast and supports JPA.

This technology stack fulfills all the stated requirements and is, therefore, a suitable foundation for the implementation of *SquatVis*. The only missing piece is a server that can run the application. For this purpose, GlassFish [Foua] is used, since it is the reference implementation for the Java Enterprise Edition [Orab].

5.3. System Design

Based on the specified requirements and the selected technologies, the architecture of *SquatVis* has been designed as illustrated in Figure 5.1. It consists of a database, the application server, and the front-end that is running in a browser. This can be seen as architecture with five layers, one for each of the three parts and two for the intermediate levels. Each of these layers only communicates with layers next to them. Thus, this ensures high internal modularity and modifiability.

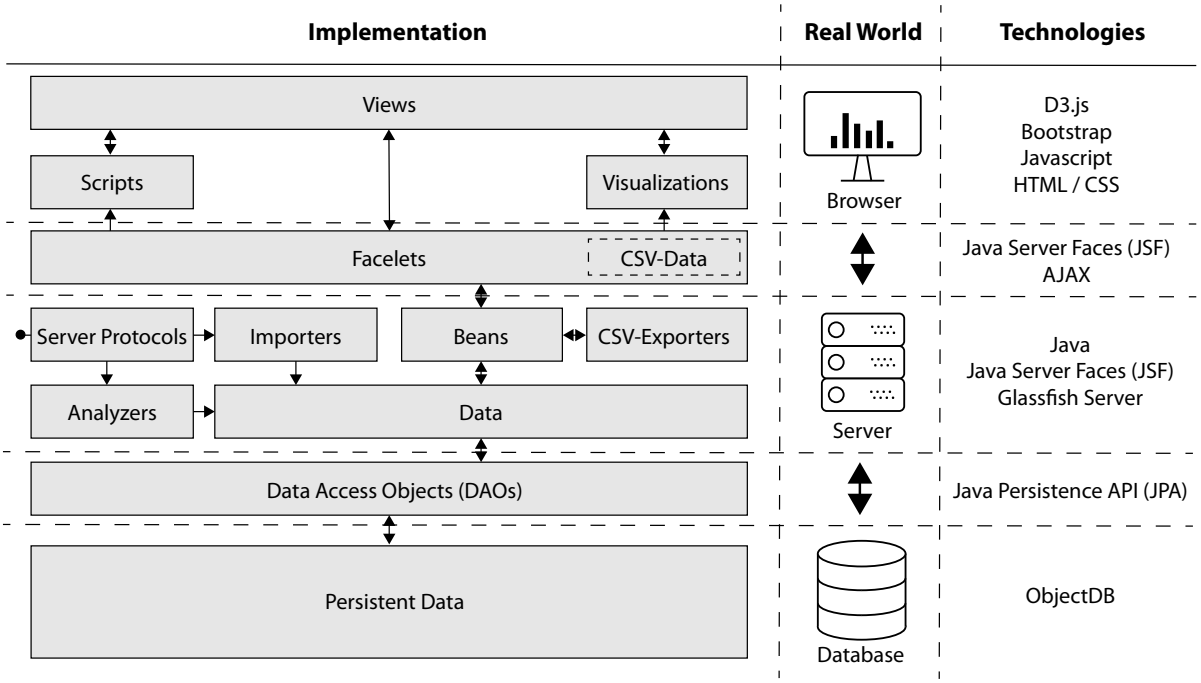

Figure 5.1.: Top-level design and technologies used in *SquatVis*.

The lowest layer is the database, which stores the data in its preferred representation. Due to the use of JPA, the database can be replaced with minimal efforts if desired. Any access to the persistent data takes place in so-called Data Access Objects (DAOs). These objects provide generic methods to add, remove, or modify the stored data. For the application layer, this data is turned into an object-oriented representation. At this point, there is no information about the visual representation present, e.g., the position of a component in a graph.

In general, the data consists of the results received from the software architecture optimization approaches and the results of the internal data analysis. The results are exchanged with these approaches by the use of a common library. *SquatVis* implements the server-side protocols which handle the exchange of the results and the necessary

communication. The received information is delegated to the importers, which transform it to fit the internal data representation of *SquatVis*. Before the protocols terminate, they trigger the analysis of the received results by analyzers. Thus, the Pareto candidates and the architectural description is delayed, so the first results can be displayed to the architects quickly.

The imported data can then be read by so-called beans. Their purpose is to answer the requests from the front-end, e.g., to export a candidate, and to manage the sessions of the architects. Thus, this layer has to store the state of the sessions and apply the transformation of the data into the required format for the front-end. The data for the visualizations are handed over to specific exporters that transform them into the comma-separated values (CSV) format. Next, the formatted data is placed by the beans in the Facelets, which also specify the structure of the views.

After a view is generated and loaded, communication with the application server is minimized. The views only notify the application server of changed settings, so these settings are kept when loading another view. However, interaction with the visualizations mostly takes place in the front-end. D3.js parses the provided, CSV-formatted data and does not need communication with the application server. Thus, the responsibility for the visual representation and its behavior is almost completely delegated to the front-end.

5.4. Data Structure

The design of the underlying data structure is an essential part of *SquatVis*. The definition of the data structure in the back-end also determines what can be visualized in the front-end. To enable the support of most of the existing software architecture optimization approaches, the design is mostly adopted from the results of the domain analysis presented in Section 4.1. An extract of the data structure is presented in Figure 5.2.

Again, the **Project** is the central class that combines all the other information. It contains a unique identifier that is also known to the associated software architecture optimization approach, which can then order *SquatVis* to update the project. In addition, information about the project's **Status** for administrative reasons is available. The purpose of the **Status** is to record the progress in the optimization process. Therefore, the progress in percent is stored, as well as status messages and time stamps of important events.

Furthermore, each project also has a **ToolConfiguration**, which describes the properties of the software architecture optimization approach that created the project. It assures that the name of this approach is known. Additionally, it holds the information about the supported features of the approach, e.g., parental relationships between candidates.

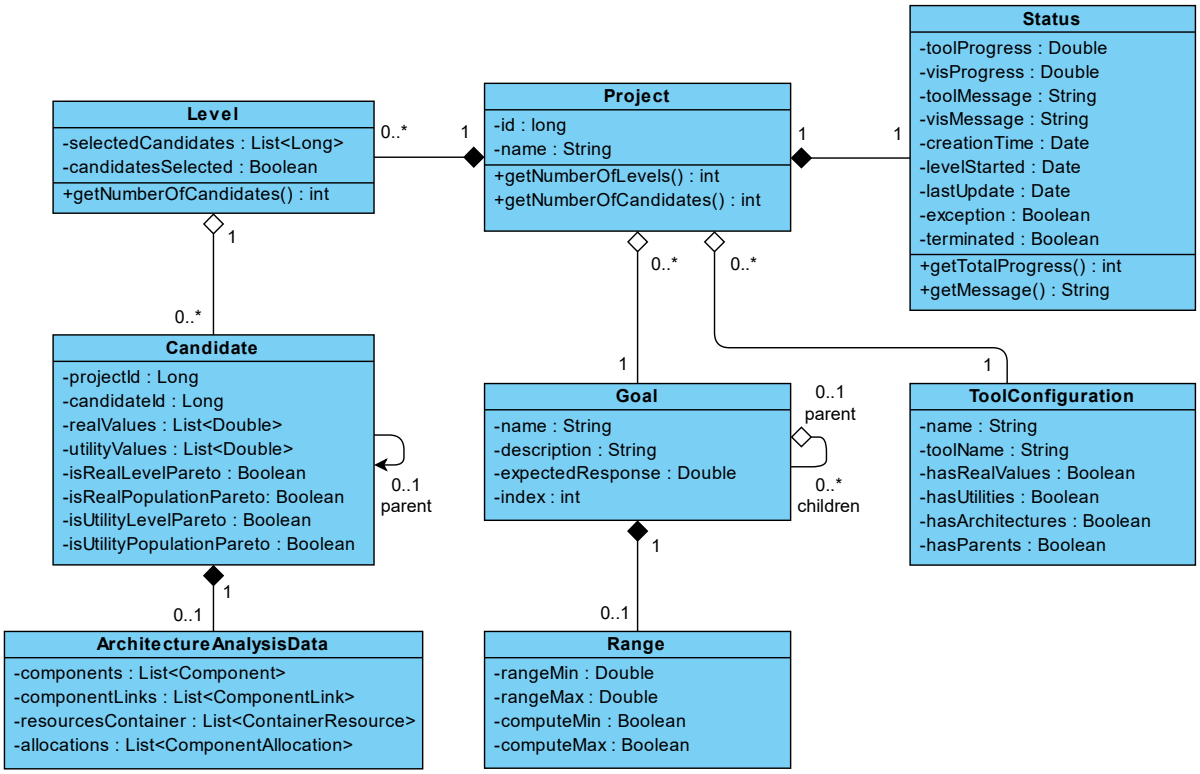

Figure 5.2.: Extract of the data structure designed for *SquatVis*.

Thus, supported features in *SquatVis* can be deactivated, so the architect can use the prototype even with degraded functionality.

Goals are also present in the data structure of *SquatVis*. Moreover, a hierarchical definition of goals is supported. This means that composite goals can be used to group atomic goals. For instance, adding a component *A* and modifying a component *B* with low effort can both be grouped as modifiability goals. However, only the satisfaction of atomic goals can be predicted by the use of metrics. For each goal, the range for its response values can be specified. As *SquatVis* only uses utility values in the visualizations, this is rather a preparation for future support of response values.

The actual proposals made by the software architecture optimization approaches are always sent as part of a whole **Level**. These levels are sorted in the order of their arrival. Each level also stores the identifiers of the selected candidates that should be used to generate the next level. These identifiers are defined by the software architecture optimization approach, so it is sufficient to send back the candidates' identifiers.

A **Candidate** holds the response and utility values that describe its satisfaction of goals. Only real values are supported and they must be in the same order as the atomic goals. In addition, the candidate also stores its result of the Pareto analysis. Finally,

an **ArchitectureAnalysisData** is attached to a candidate, as soon as the analysis is finished. These analysis results consist of lists of components, component links, resource containers, and allocations. The actual structure of the analysis data is identical to the definition of the architectural model from Section 4.3.2.

5.5. Connector Interface

The so-called Connector Interface is a common library that allows the exchange of information between *SquatVis* (server-side) and software architecture optimization approaches (client-side). It provides an own data structure, which is a subset of the representation presented in Section 5.4. The difference is, that these data types must be transferable by the use of Java Sockets. In addition, client-side protocols are provided. Thus, developers need to do two things to add support for *SquatVis* to their tools: On the one hand, they have to transform their own representation of their results into the representation specified in the library. On the other hand, they need to hand over these results to the right protocols at the right time. The remote communication is then handled automatically.

Since the whole Connector Interface is written in Java, the client-side must also use Java. If the software architecture optimization approach is written in another programming language, there is still the option to use another technology that can be used to transfer the data to a Java program, e.g., by exporting the results in a textual form to the HDD. Thus, the use of Java is not a too strong restriction. Furthermore, a Java program can even be used to load existing data of an already terminated optimization process to investigate the results with *SquatVis*.

The communication protocols are defined based on the lifecycle of a project, which is outlined in Figure 5.3. A client-side and a server-side protocol exists for each of the displayed activities. At the start of the optimization, the creation of a new project has to be requested. In this process, some decisions have already been made, e.g., the goals for the optimization are defined before the optimization starts. Then, the optimization begins. Depending on the project and the actual software architecture optimization approach, the optimization can run for hours or even days. To signal the architect that the optimization is still running, progress updates can be sent. Although this is an optional step, it prevents *SquatVis* from assuming that the optimization timed out.

The optimization can be interrupted to ask the architect for her preferences, to send intermediate results, or it simply terminates. In all cases, the client needs to send a level of candidates to the visualization server. If the optimization terminated, the client can instantly decide to also terminate the project. By doing so, the visualization server can

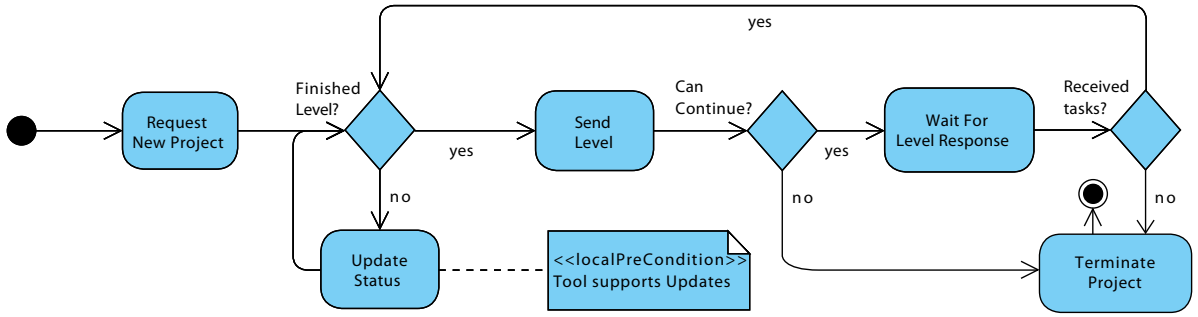

Figure 5.3.: Activity diagram for the lifecycle of a project from the client's perspective.

signal the architect that no interaction is required. If the optimization can be continued, then the client should wait for a response by the visualization server. This response specifies the candidates that should be used to generate the next level of search. It has to be noted that these candidates need to be selected by the architect. Thus, some time will usually pass until a response is received. In the case that the provided response is empty, the client can usually not proceed with the optimization. Therefore, there is again the option to terminate the project. In the other case, the optimization proceeds and a new level is generated.

Figure 5.4 outlines the behavior of the client and the server in more detail for a new project request. On the client-side, a project, the client's configuration, and the optimization goals are known and handed over to the **NewProjectClientProtocol**. This protocol is then executed on the client-side. It uses a Java socket to send a message over the network. On the server side, the **ConnectorServer** is continuously listening to such requests. As soon as the request is received, the server starts a **ConnectorRequestHandler** in a new thread and waits for other requests. A **ServerProtocolDispatcher** is then asked to provide the correct server-side protocol to answer the request. In this case, a **NewProjectServerProtocol** is returned and initialized. The protocol retrieves the sent project, configuration, and goals. They are handed over to the appropriate importers, are processed and stored into the database. After all data has been received, the server-side transmits the project identifier to allow the client-side to identify and update the created project.

At this point, the client-side protocol terminates. The server-side, however, continues. The request-specific **NewProjectPostProtocolHandler** is initialized and returned to the **ConnectorRequestHandler**. It is responsible for running protocol-specific tasks after the protocol's termination, e.g., updating the project status or starting the analysis of the received results. As a new project does not require a Pareto analysis or an architectural

analysis, nothing is happening in this example. The general behavior for the other protocols is similar to the one in the described example.

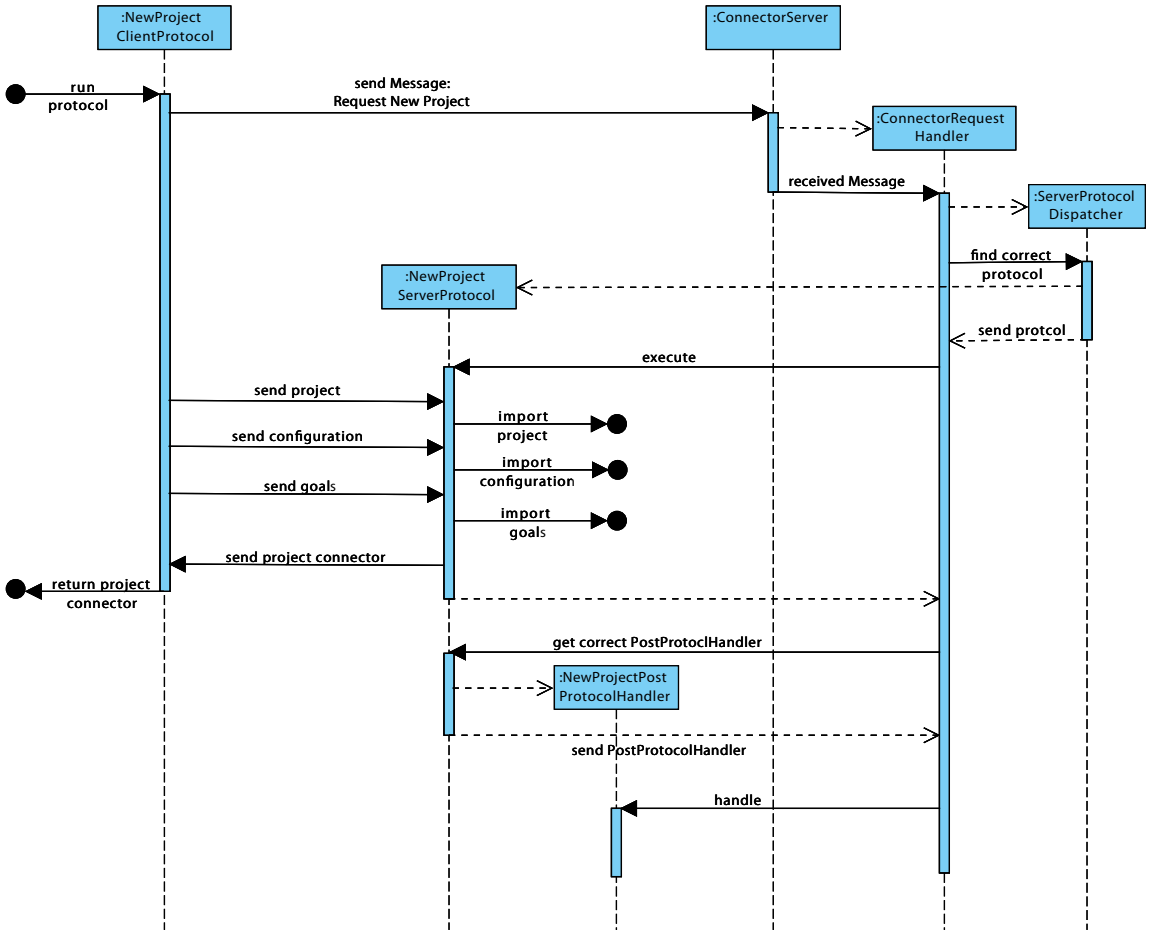

Figure 5.4.: Sequence diagram for the request of a new project.

5.6. Views

The views introduced in Section 4.6 have mostly been implemented as proposed. At some points, reasonable adjustments have been made due to practical considerations and the domain experts' feedback. A major difference, however, is that the Candidate View and the Goal View are not part of the initial version of *SquatVis*. As explained before, this decision was made because they are not as vital as the other views and to provide a working prototype faster. Another major difference is the extension by two administrative views, namely the Projects View and the Project View, which are examined

in Section 5.6.1. Then, the differences to the initial drafts of the views are mentioned for the Population View (see Section 5.6.2), the Candidates View (see Section 5.6.3), and the Architecture View (see Section 5.6.4).

5.6.1. Administrative Views

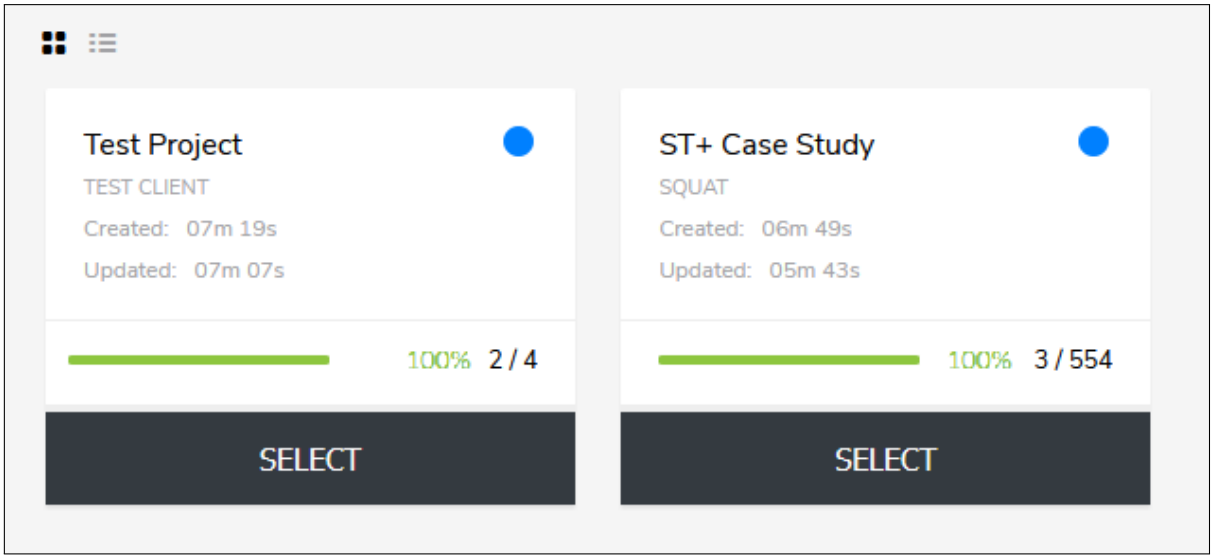

Figure 5.5.: Projects View of *SquatVis* showing two selectable projects.

The **Projects View** is shown in Figure 5.5. It is the first visible view after starting the application. Each card represents one of the available projects. At the top of the card, the names of the project and the optimization tool which delivered the data are displayed. For example, the project on the right contains the data of the ST+ Case Study, which was generated by SQuAT. Below, the date of creation and the last update are displayed.

While the progress bar displays the progress of the software architecture optimization approach that is generating the next level, the colored circle indicates the overall project status. In the example, the status is blue, which means that the optimization tool terminated and no further data will be sent or received. There is also green for running optimization and red for an error or timeout. Finally, yellow indicates that interaction is required to select candidates for the next level. To give a first impression of the loaded data, the number of available levels and candidates in this project are displayed to the right of the progress bar. The ST+ Case Study consists of three levels, including the initial level, and 554 candidates. The project can be selected by clicking at the “SELECT”-button, which loads the project and opens the Project View.

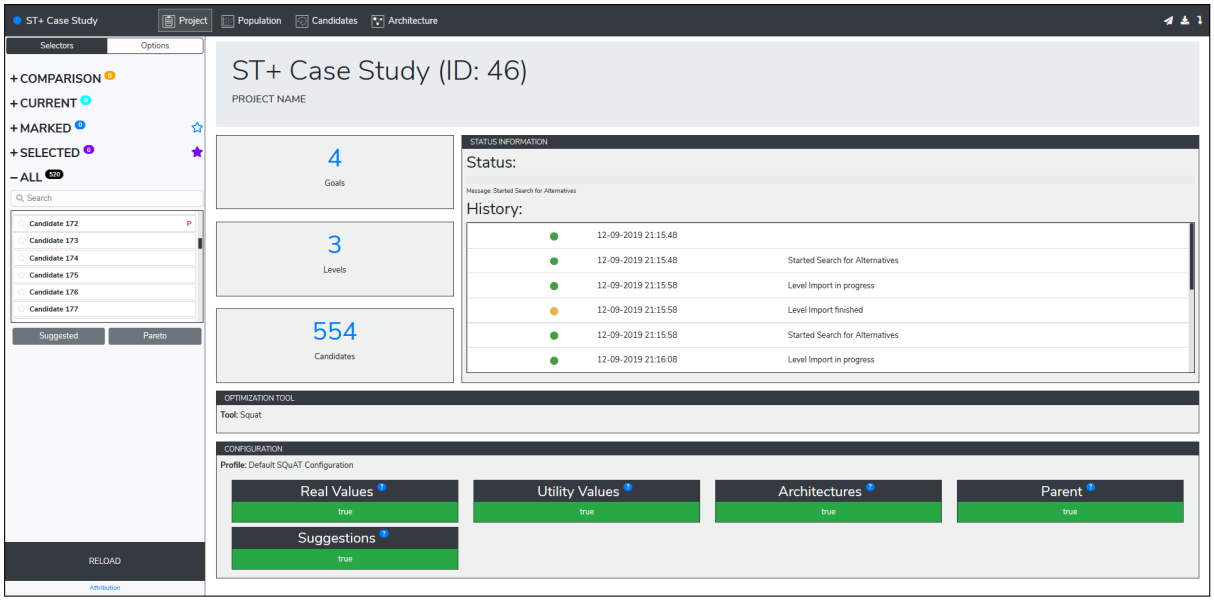

Figure 5.6.: Project View of *SquatVis* showing the ST+ Case Study project.

The **Project View** is shown in Figure 5.6. Like all project-specific views, it has a header at the top. In the header, the project status and name are shown. Besides, it is possible to edit the project name here. The header also allows navigating to the other views within the project. Buttons to export or send *selected* candidates, as well as a button to leave the project, are positioned at the right side of the header.

The toolbar on the left side is implemented as presented in Section 4.6.1. In addition, it has been extended with a reload button. The reason for this is that the visualizations need to be reloaded when the global options are altered.

The Project View itself provides more detailed information on the project than the Projects View. It contains the progress bar, provides status messages and a status log. On the left, the current number of goals, levels, and candidates in the project are displayed. At the bottom, the project's configuration is shown. This allows the architect to see which features are active in this project.

5.6.2. Population View

The **Population View** is displayed in Figure 5.7. An obvious difference to the conceptual design presented in Section 4.6.2 is that only the upper half of the matrix is displayed. The reason for this decision is that the bottom half of the matrix is just a mirrored copy of the upper half. Thus, it does not provide much value to the view. However, removing

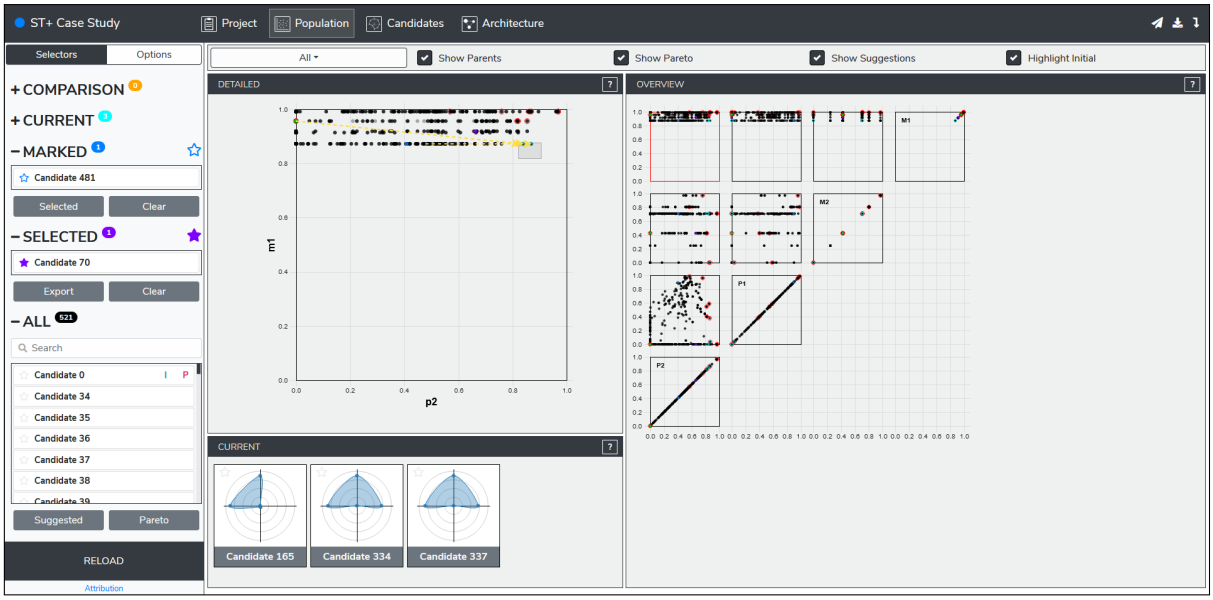

Figure 5.7.: Population View of *SquatVis* showing level 2 of the ST+ Case Study.

the bottom half of the matrix reduces the number of elements in the view and, therefore, the loading time of the page.

The other changes to the Population View are rather small. The controls are moved to a toolbar at the top, as they affect the matrix and the magnified scatter plot to the left. In addition, more options for tag control is added to this toolbar. In contrast to that, the option to toggle the population in the radar charts has been deactivated, as this noticeably increased the loading time of the view. Population data can still be viewed in the Candidates View.

5.6.3. Candidates View

Minor changes have also been made to the **Candidates View**, which is illustrated in Figure 5.8. The lines in the radar chart have been replaced with curves, mainly for aesthetic reasons. Additionally, an option has been added to customize the degree of opacity for the candidates in the background. This is due to the fact that it is hard to define a default value that is appropriate for levels of all sizes. For the same settings, a level with few candidates could always look bright, while a level with many candidates could easily darken the whole view. Thus, the architect can adjust the opacity to the size and values in the level.

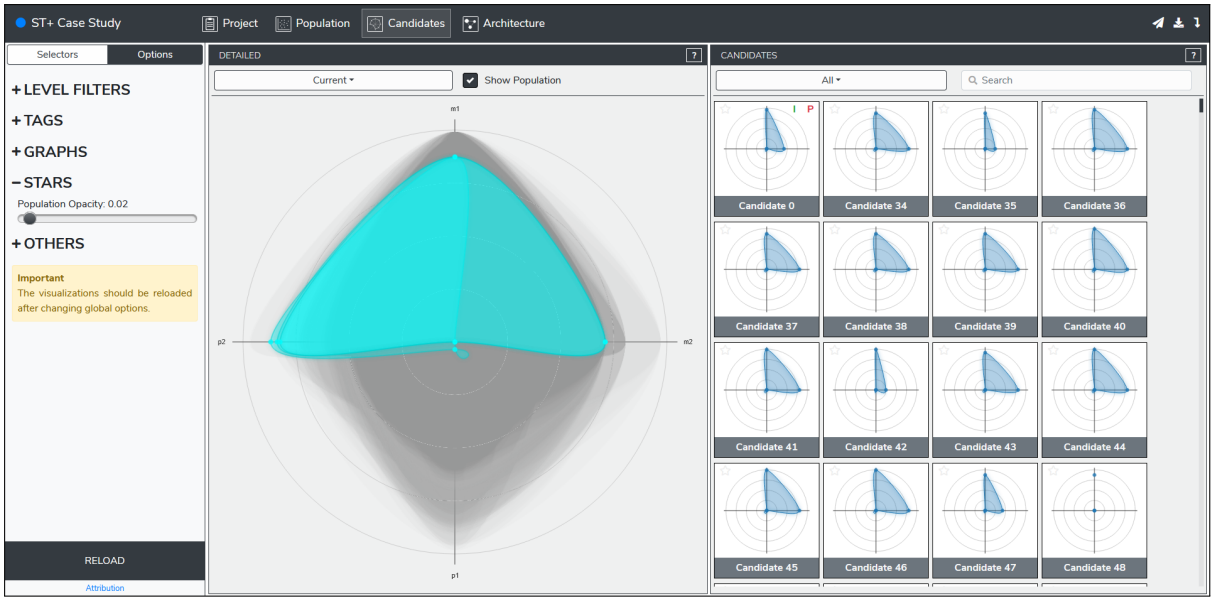

Figure 5.8.: Candidates View of *SquatVis* showing level 2 of the ST+ Case Study.

5.6.4. Architecture View

The implementation of the **Architecture View** is also quite close to the proposed design presented in Section 4.6.3. For consistency with the graph, the parallel coordinate plot also displays candidates which are not grouped as *current*. The “Reduce Graph”-option is moved to the view’s toolbar at the top. In addition, to simplify the implementation, the initial implementation for the graphs does not allow for zooming.

Several options for controlling the graphs have been added, mainly as assistance to mitigate visual cluttering in the graphs. The architect can customize the length and strength of both graphs. In addition, filters for elements used by a few and/or many candidates are added. These filters are applied in addition to the graph reduction, which emphasizes the differences in the graphs. Besides, the displayed names of the components can be shortened.

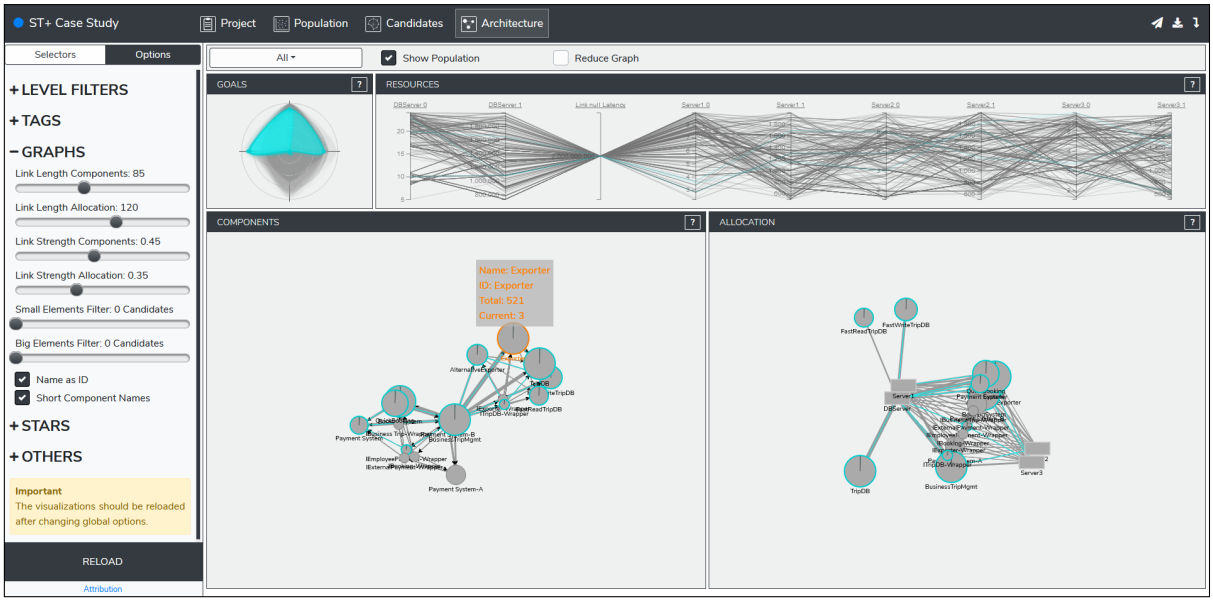

Figure 5.9.: Architecture View of *SquatVis* showing level 2 of the ST+ Case Study.

Chapter 6

Evaluation

In this chapter, the visualization approach that has been introduced in Chapter 4 is evaluated. The evaluation is based on the prototypical implementation that was introduced in Chapter 5. First, Section 6.1 defines the goals of the evaluation. In Section 6.2, the methodology for the evaluation is outlined with respect to the defined goals. Two case studies, which are used in the evaluation, are introduced in Section 6.3, before the experimental setting is described in Section 6.4. Finally, Section 6.5 summarizes the results of the evaluation, while Section 6.6 interprets and discusses these results.

6.1. Evaluation Goals

To evaluate the proposed approach, the goals for the evaluation have to be defined. For the evaluation of this work, three research questions are formulated. The first research question aims at the overall validation of the approach. The second one is concerned with the scalability of the approach with regard to the amount of visualized data. As this is the only visualization approach that is explicitly focusing on the domain of software architecture optimization, it is expected that more improvements have to be made until the approach becomes mature. Thus, the last research question focuses on suggestions for future work.

These research questions are formulated as follows:

- **RQ1:** Is the *SquatVis* approach able to assist architects in software architecture optimization?
- **RQ2:** How scalable is the *SquatVis* approach?
- **RQ3:** How can the *SquatVis* prototype be improved?

In the following, these research questions are examined in more detail. Additionally, they get refined and split into subgoals. Section 6.1.1 is concerned with functionality (RQ1), while Section 6.1.2 focuses on Scalability (RQ2). Then, Section 6.1.3 addresses the identification of improvements (RQ3).

6.1.1. Functionality

The first goal is to validate the proposed visualization approach. Assuring that the visualizations and views technically fulfill their specifications can easily be done by visual observation. However, the approach must also fulfill its overall purpose to assist architects in the domain of software architecture optimization. Therefore, it should be investigated whether it sufficiently supports the typical use cases experienced by the architects. In Section 4.2, four essential use cases for the domain of software architecture optimization have been identified. To fulfill its purpose, the proposed approach must be able to deliver satisfying results for at least one of these use cases, ideally for all the use cases. Therefore, the first research question can be refined, based on these use cases, as follows:

- **RQ1.1:** Can the *SquatVis* approach assist architects in selecting candidates for the next level?
- **RQ1.2:** Can the *SquatVis* approach assist architects in the decision of whether to stop the optimization?
- **RQ1.3:** Can the *SquatVis* approach assist architects in implementing a selected candidate?
- **RQ1.4:** Can the *SquatVis* approach assist architects in exploring and explaining results?

6.1.2. Scalability

Scalability is an important property for the proposed approach, as scalability directly influences its usefulness in practice. A software architecture optimization approach can send many results, which need to be handled by the visualization approach. Therefore, it is important to know where the boundaries for the size of these results lie. However, this rather requires a rough estimation of the result's size than an exact number.

Scalability can be investigated from two different perspectives. From the technical perspective, it can be examined which result sizes can still be handled by an actual implementation of the approach without significant degradation of functionality. From

the conceptual perspective, the amount of information that a human can handle when using the approach can be investigated. The difference is that the approach might be able to present the results, but the architect can not interpret them anymore, e.g., because of too much visual clutter. In this case, the approach can not fulfill its purpose anymore. Thus, the second research question can be split:

- **RQ2.1:** How scalable is the *SquatVis* approach with regard to the technical possibilities?
- **RQ2.2:** How scalable is the *SquatVis* approach with regard to the architect's capabilities?

6.1.3. Improvements

The research question regarding improvements emphasizes that the proposed approach is a first, experimental step in providing visual assistance for architects in software architecture optimization. As stated before, the satisfaction of the architects' expectations is bound to be subjective. It is, therefore, unlikely that the proposed approach can fulfill all, sometimes even conflicting, expectations of the architects. Thus, the continuous evolution of the approach through user feedback is necessary and this research question should determine the direction of future development.

It also needs to be considered that the Goal View and the Candidate View have not been implemented. Thus, it should be evaluated whether such views are required or not. Additional visualizations and views could be added. The question for improvements is deliberately formulated openly, so the architects, who use the approach, can decide what is necessary to increase their satisfaction.

6.2. Evaluation Methodology

The evaluation is divided into three parts. First, experts had the chance to provide feedback during the development process as outlined in Section 6.2.1. Most of the stated research questions require a qualitative evaluation approach through expert reviews, which are explained in Section 6.2.2. Other research questions can be answered by a quantitative analysis of performance measurements. This method is outlined in Section 6.2.3.

6.2.1. Pre-Evaluation Interviews

To include the domain experts' feedback earlier in the process, paper prototypes [Sny03] have been designed. Then, domain experts viewed these prototypes and were asked to provide feedback. In a later stage, a first stable version of the prototype has been presented to the domain experts. The experts were asked for critique and potential improvements, again. Although this is not a sufficient method to evaluate this work, it still helps to justify design decisions, especially when they are proposed by the domain experts. For this reason, the outcome of these pre-evaluation interviews is also reported.

6.2.2. Expert Reviews

The research questions RQ1.1 to RQ1.4 all require the participation of humans in the evaluation, as the most natural way to judge whether the approach can assist architects is to ask the architects for their judgment. The same holds for RQ2.2, which requires the architects to judge their capabilities. However, in the design of the evaluation, also the outer circumstances have to be taken into account. Understanding software architecture optimization requires some expertise and the number of practitioners and experts in this domain is limited. Also, no other established visualization approaches are focusing on the domain of software architecture optimization, yet. This makes the design of a quantitative evaluation difficult, as usually many participants are required.

However, a quantitative evaluation is not necessarily required for this work. Greenberg and Buxton [GB08] even describe such techniques as harmful in early design phases and for academic prototypes because creative ideas can be killed and meaningful critique is often not delivered. Isenberg et al. [Ise+13] also emphasize the need for qualitative evaluation with real users and their problems. In their work, they reviewed the practice of evaluating visualization based on publications at the IEEE Visualization conference and the IEEE Information Visualization conference. The results show that some publications evaluate newly introduced visualization tools by assessing their value for domain experts. This assessment is usually performed by case studies, which represent usual problems faced by domain experts. The results also indicate that the popularity of such a kind of evaluation increased in the last years.

Isenberg et al. [Ise+13] mention that a number of "three or five [participants] is very convincing, however, often even 1-2 is sufficient, if interesting conclusions can be derived and if the study is comprehensive and detailed". Additionally, such an evaluation is usually applied when the problem description is fuzzy and low-level tasks are hard to derive. This is the case for this work. As an example, Xu et al. [Xu+19] used this kind

of evaluation by case study with only two domain experts to evaluate their approach for the interactive analysis of performance issues in cloud computing. Another example is the work of Lundstrom et al. [Lun+11], which describes interviews with five domain experts to understand multi-touch visualizations.

Thus, this work uses the same method for the evaluation. Domain experts review the tool by solving tasks on case studies. Each task refers to one of the four use cases. These tasks are accompanied by a catalog of questions, which allow answering the research questions on the functional usefulness (RQ1.1 to RQ1.4), human-centered scalability (RQ 2.2), and potential improvements (RQ3).

6.2.3. Performance Measurements

RQ2.1 aims at the scalability of the approach from the technical point of view. Measurements on the prototypical implementation can be applied to evaluate the scalability of the approach in practice. Therefore, the full loading times for the three implemented visualization views Population View, Candidates View, and Architecture View can be investigated. Although the loading time of the view is not the only possible metric to measure scalability, it is a simple metric, easy to measure by various tools, and has a direct influence on the architects' user experience.

Various independent variables could be considered, e.g., the number of candidates, goals, or architectural components. However, the number of goals in a project is difficult to control without the Goals View. Architectural entities are also hard to control, especially when a case study system is investigated and no synthetic data should be used. However, *SquatVis* allows to filter levels. The active levels also determine the number of visible candidates. Additionally, the number of candidates is of practical relevance, as a software architecture optimization approach might propose many of them. Thus, this evaluation focuses on the investigation of scalability with regard to the number of candidates.

A threshold for the acceptable loading time also needs to be defined to determine the maximum number of candidates based on the measurements. In the web, page load times under two 2 seconds are considered desirable to meet the users' expectation [Nah04]. While this would allow for quick switching of views, worse values could also be acceptable for different reasons. First, the described approach is not a website. Second, the approach is used by experts, who are willing to wait to get meaningful results. Finally, switching views does usually not happen often. Thus, a loading time of 10 seconds is assumed to be sufficient in practice. If switching of views is minimized, even a loading time of 60 seconds could still be acceptable, although this would result in reduced usefulness for the architect. However, this is not much time, if the architect gets insights, which she would not get without the approach.

6.3. Case Study Systems

There are currently two case study systems available for SQuAT [Rag+17]. The first one is Simple Tactics Plus (ST+), which is outlined in Section 6.3.1. The second one is the Common Community Modeling Example (CoCoME), which is explained in Section 6.3.2.

For both case studies, the SQuAT [Rag+17] approach has been used to optimize an initial architecture. For this purpose, performance bots [Fra16] and modifiability bots have been utilized. While the performance bots have been instructed to predict the system's response time, the modifiability bots have been used to predict a complexity measure, which is based on cyclomatic complexity [McC76].

To reach their goals, all bots have been instructed to use all their available transformations. Thus, the performance bots have been allowed to replace components with others and to allocate components to arbitrary servers. Additionally, modifying hardware resources was allowed, too. The only modifiable hardware resources have been the clock speed of CPUs and the speed of HDDs at the servers. The modifiability bots have been equipped with so-called wrappers and splitters. Wrappers create a new component, which handles all the outgoing dependencies of another component. Therefore, all these dependencies are handled by the created component. Splitters, on the other hand, split a component into two components if two different interfaces are provided.

Name	**ST+**	**CoCoME**
# Goals	4	8
# Basic Component Types	5	51
# Alternative Components Types	4	4
# Levels	2	5
# Candidates	554	1193

Table 6.1.: Key indicators for the two case study system ST+ and CoCoME.

The case studies are presented in more detail in the following. Table 6.1 summarizes some of their key indicators. In conclusion, it can be said that the CoCoME Case Study is noticeably more complex with regard to the problem specification, as well as the generated solution space. There are twice as many goals specified and the initial architecture contains ten times more used components. More than twice as many solutions are proposed, while the optimization reached level 5, in contrast to level 2 for the ST+ Case Study.

6.3.1. ST+ Case Study

Simple Tactics was originally published as part of the architecture optimization approach PerOpteryx [KR11] as an example system. It models a business trip management system, that can be used to book and pay business trips. The original model was created for optimization with regard to performance, reliability, and cost. This model has been extended in the context of the SQuAT [Rag+17] approach as a suitable case study system for the evaluation of SQuAT [Rag+17] with regard to the quality attributes performance and modifiability. Therefore, the initial model has been extended with a database to store business trip information and an exporter, which provides this information in a human-readable, printable form. In addition, also alternative components for the exporter and database have been defined. The resulting initial model is the so-called Simple Tactics Plus (ST+) system, which has been used multiple times in projects in the context of SQuAT [Fra16; PRF17; Rag+17]. This model is also publicly available [Frab; STb].

For the optimization of the initial architecture, four goals are defined in the scenario format [CKK02]. The two scenarios **p1** and **p2** aim at improving the system's performance, while the other two scenarios **m1** and **m2** aim at improving its modifiability:

- **p1**: This scenario assumes a slight increase in the number of users in the future. In the case of an overall system's usage increase of 10%, the system still needs to respond within 30 milliseconds.
- **p2**: One of the two servers in the database cluster becomes unavailable due to a failure or crash. In this case, the system still has to respond within 40 milliseconds.
- **m1**: A new payment option is added to the system. The necessary changes to the system should not have a complexity value of 120 or higher.
- **m2**: A security token to database transactions is added to the system. The necessary complexity to apply this change should be less than 300.

Level i	**0**	**1**	**2**
Candidates in Level i	1	33	520
Candidates until Level i	1	34	554

Table 6.2.: Number of candidates for each level in the ST+ Case Study. The initial architecture is denoted as level 0.

Two levels of search have been conducted based on the described setting. As shown in Table 6.2, the number of candidates increased significantly with each level. While the first level only consisted of 33 candidates, the second level already consists of 520

candidates. This is due to the fact that the whole first level was selected as the base population to generate the second level. Figure 6.1 shows the evolution of the levels based on the visualization in the Candidates View of *SquatVis*. With each level, more area seems to be covered. This indicates that improvements have been made with regard to the utility values.

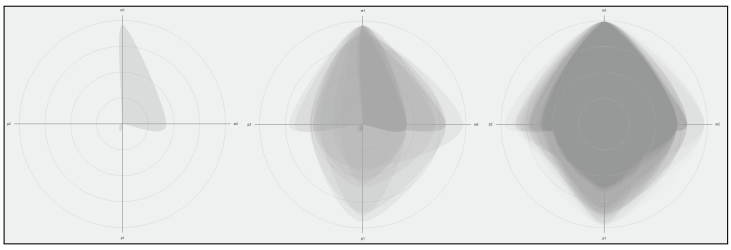

Figure 6.1.: Evolution of utility values in the ST+ Case Study. Each radar chart represents one level and is taken from the Candidates View. The charts are in ascending order, starting with the initial level at the left. Goals are ordered clockwise, starting at the top: $m1$, $m2$, $p1$, $p2$.

It has to be noted that the settings of the optimization differed from the currently applied settings in SQuAT [Rag+17]. The prototype did not log the suggested candidates at the time when the case study dataset was created, so no suggestions are available. In addition, overoptimization has been penalized by the used utility function. This means that the optimization preferred values close to the expected response, while actually "better" response values have utility values below one. Still, the underlying concept did not change. Thus, this case study dataset is still of interest for this work.

Figure 6.2 presents the radar charts of the nine found Pareto candidates in level 2 of the ST+ Case Study. It can be seen that there is no candidate that fully satisfies the goals that have been stated for the optimization. However, *Candidate 280* seems to be more balanced than the others. Therefore, architects need to make trade-offs. Besides, the figures in Section 5.6 also presented the actual implementation of the view using the ST+ Case Study results. Figure 5.7 visualizes level two of the ST+ Case Study for the Population View, Figure 5.8 for the Candidates View, and Figure 5.9 for the Architecture View.

The matrix in the Population View (see Figure 5.7) shows some patterns for level two. For example, the scatter plots related to modifiability goals contain many circles forming lines. This indicates that many changes did not influence the modifiability goals, as their response stayed the same. Another example is that the circles for $m1$ are all located in the upper quarter of the plots. Thus, the goal is always (almost) satisfied, regardless of the selected architecture.

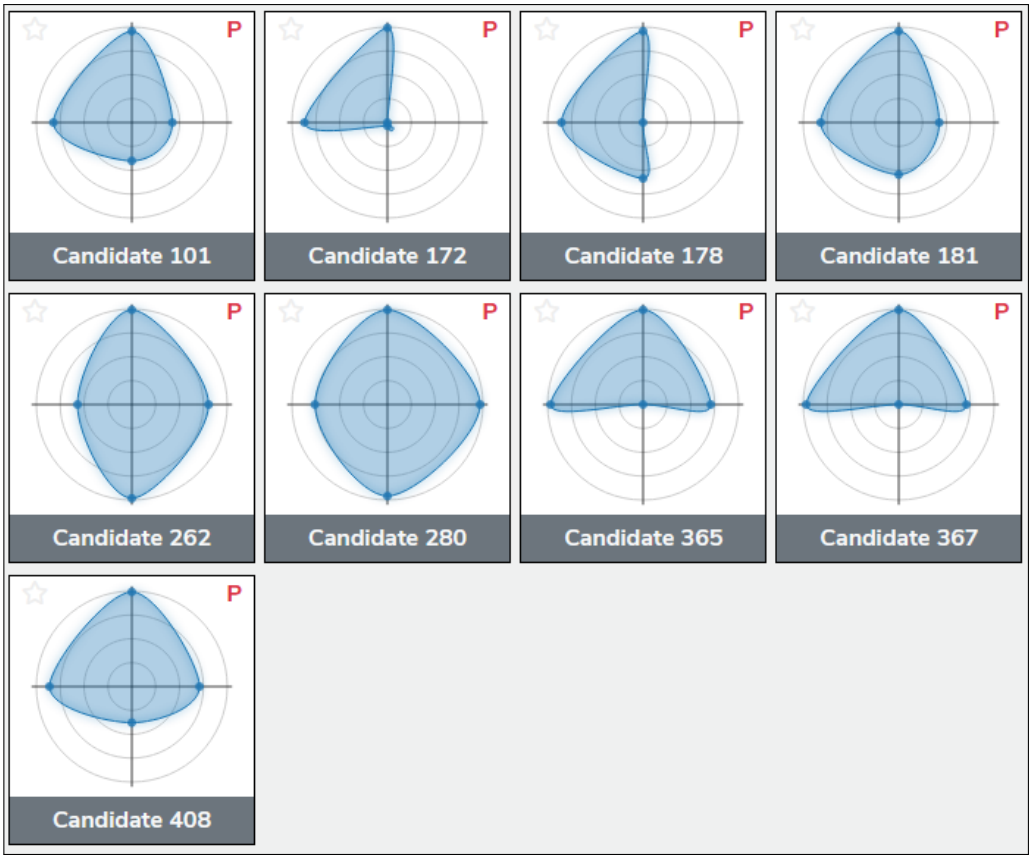

Figure 6.2.: Radar charts showing the Pareto candidates in the second level of the ST+ Case Study.

The Architecture View (see Figure 5.9) also provides some points of interest. The initial architecture contains only five components and four alternative components. The graph, however, contains more than these nine components. Additionally, there are components that are used more rarely, as there are some small circles. Thus, the modifiability bot used wrappers and splitters to create new components. The four initial servers in the initial architecture are preserved and the performance bots allocated the components to different servers, as indicated by the several outgoing edges in the visualization for allocation. Only the database components are solely allocated to the database server, as indicated by their single edge. According to the resources visualization, each server contains two resources. The first one for each server is the CPU, the second one the HDD. Although the resources' values seem to vary, a pattern is visible: It appears like there are more often high values for the CPUs, while the values for the HDDs are rather low. This could indicate, that the CPU is a kind of bottleneck for the system, while the HDD is not.

6.3.2. CoCoME Case Study

The Common Community Modeling Example [Rau+08] (CoCoME) has originally been designed for comparison and validation of modeling approaches in the context of component-based systems. It has been extended as part of the DFG Priority Programme "Design for Future — Managed Software Evolution" [Bou+19]. Thus, the model can be seen as a benchmark for modeling approaches. It models the trading system of a supermarket with a lot of requirements and components, e.g., barcode scanners, cash desks, and displays. An implementation of the model also exists for PCM [Tec]. In SQuAT [Rag+17], this model has been adjusted to serve as case study to evaluate the approach. The reason for this is that it is assumed to be a realistic, large scale system. In addition, it could be used to compare SQuAT [Rag+17] to other software architecture optimization approaches. Due to the technical requirements of the bots, composite components have been removed and events had to be replaced. Also, alternative components have been added. Additionally, user behavior and the hardware level had to be modeled for the use in SQuAT [Rag+17]. Still, the system is semantically similar to the original system. This modified model is also publicly available [Frab; STa].

For CoCoME, eight goals are defined in the form of scenarios [CKK02]. The scenarios **p1**-**p4** are related to performance, while the scenarios **m1**-**m4** focus on modifiability:

- **p1**: This scenario assumes a slight increase in the number of users in the future. In the case of an overall system's usage increase of 10%, the system still needs to respond within 1.2 seconds.
- **p2**: In this scenario, a big increase in the number of users is expected in the future. If the overall system's usage increase is 50%, the system has to respond in 1.4 seconds or less.
- **p3**: One of two servers in a cluster can become unavailable due to a failure or crash. In this case, the system still has to respond within 1.0 seconds.
- **p4**: This scenario assumes a change of customer behavior, e.g., during sales. If the customers buy more items in average, the system must respond within 2.4 seconds.
- **m1**: A new payment option based on near field communication (NFC) is added to the system. The necessary changes to the system should not have a complexity value higher than 2270.
- **m2**: A cash desk service for premium customers should be added. The necessary complexity to apply the changes should be less than 750.
- **m3**: A service to withdraw money at a cash desk should be added. A complexity of less than 170 is desired for the implementation of the necessary changes.

- **m4**: A logging service for the system should be added. The required changes must have a complexity of less than 2180.

Level i	**0**	**1**	**2**	**3**	**4**	**5**
Candidates in Level i	1	125	344	301	120	302
Candidates until Level i	1	126	470	771	891	1193

Table 6.3.: Number of candidates for each level in the CoCoME Case Study. The initial architecture is denoted as level 0.

The initial architecture has been optimized by SQuAT, as described before, and five levels of search have been conducted. Table 6.3 shows the numbers of candidates at each level. In contrast to the ST+ Case Study, only a selection of candidates at each level has been used to generate the next level. Each bot was ordered to rank the candidates according to its satisfaction and to propose the highest ranked candidates as seeds for the next level. Thus, the number of candidates in each level remained comparatively small, even though the CoCoME system model itself is larger than the ST+ system model.

Figure 6.3 shows the evolution of the level with regard to utility values as visualized in the Candidates View. It can be seen that the shape does not change much after level two. Therefore, it can be assumed that the effect of the made improvements was smaller. In the higher levels, almost the whole space is covered. This indicates that the goals, in general, are satisfiable, although it is not known whether this also holds for particular candidates.

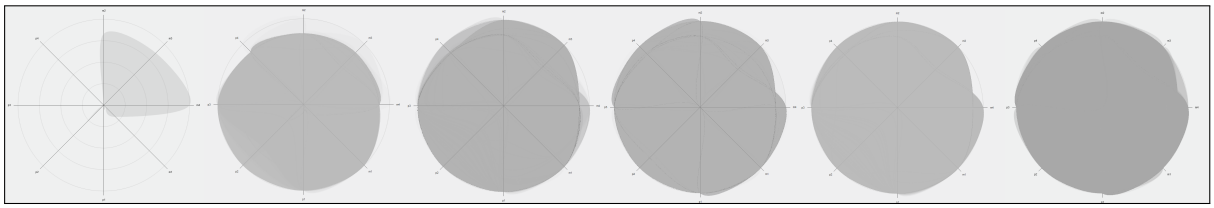

Figure 6.3.: Evolution of utility values in the CoCoME Case Study. Each radar chart represents one level and is taken from the Candidates View. The charts in ascending order, starting with the initial level at the left. Goals are ordered clockwise, starting at the top: $m2$, $m3$, $m4$, $m1$, $p1$, $p2$, $p3$, $p4$.

The optimization has been performed with the default settings of SQuAT [Rag+17]. Thus, suggestions are logged and can be presented to the architect. Overoptimization is not penalized, which means that every “better” response value than the expected response value has a utility value of exactly one. Still, a far better solution is not preferred over another that only barely satisfies a goal, but it is not penalized anymore.

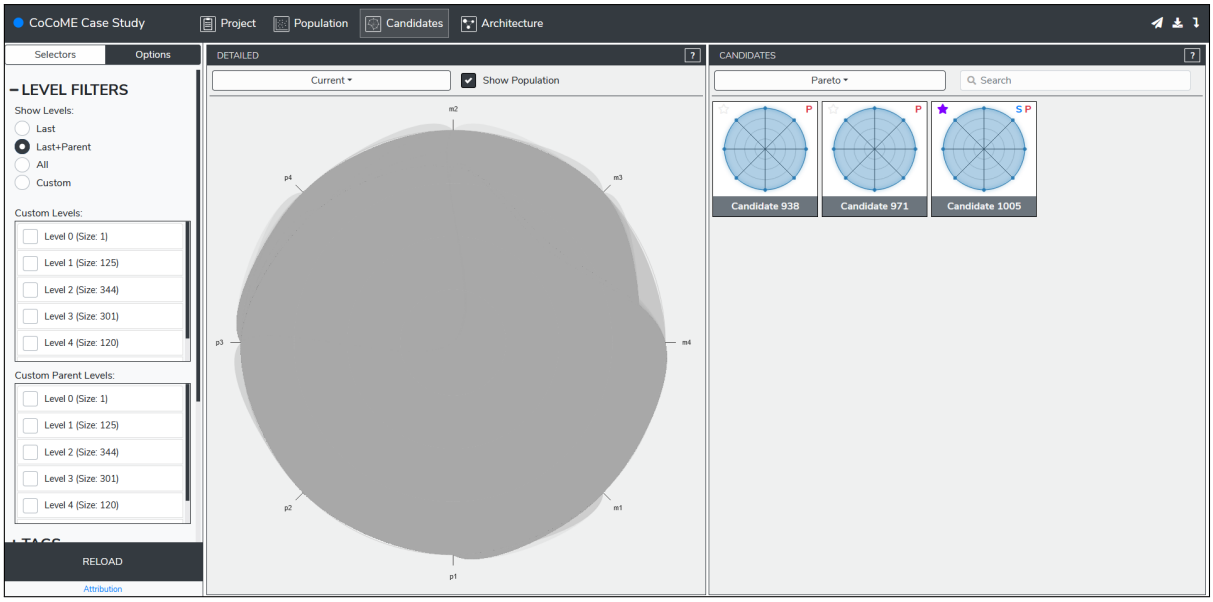

Figure 6.4.: Candidates View of *SquatVis* showing level 5 of the CoCoME Case Study. Goals in all radar charts are ordered clockwise, starting at the top: $m2$, $m3$, $m4$, $m1$, $p1$, $p2$, $p3$, $p4$.

Figure 6.4 shows the Candidates View displaying the fifth level of the CoCoME Case Study. The Pareto candidates of this level are displayed at the right. These three candidates satisfy all the eight goals. Therefore, they can be interpreted as ideal solutions. It has to be noted that the last of these candidates was also suggested by SQuAT [Rag+17].

The Population View for the fifth level of the CoCoME Case Study is shown in Figure 6.5. It is noteworthy that the circles seem to be highly clustered. Many of the circles are also rather close to the top right corner. This can have several reasons, e.g., most of the candidates are already ideal or close to being ideal. It could also be assumed that the applied transformations had only a small effect on the utility values at this level. Anyways, these patterns are different from the ones observed in the ST+ Case Study.

The architecture of the initial candidate in the CoCoME Case Study is significantly larger than the one of the ST+ Case Study. The initial architecture consists of 51 components and 5 alternative components. However, the number of servers still is four and HDDs are not modeled. Therefore, only the four CPUs and the latency of the network are present in the visualization of resources in the Architecture View (see Figure 6.6). It is noteworthy that there are only three existing configurations for the resources. The reason might be that the seed candidates for this level all have been last modified by performance at the previous level. In this case, the performance bots do not optimize these candidates again at this level. This would mean that resource values and the allocations are not changed, as the modifiability bots do not modify these architectural properties. This assumption

is affirmed by a noticeable number of candidates which are always allocated at the same server. However, it can be seen that there are more displayed components than in the initial architecture. Besides, some circles are small. This means they are used rarely, which indicates that they have been generated specifically for one candidate. Thus, the modifiability bots obviously applied its transformations and created wrappers and/or split components.

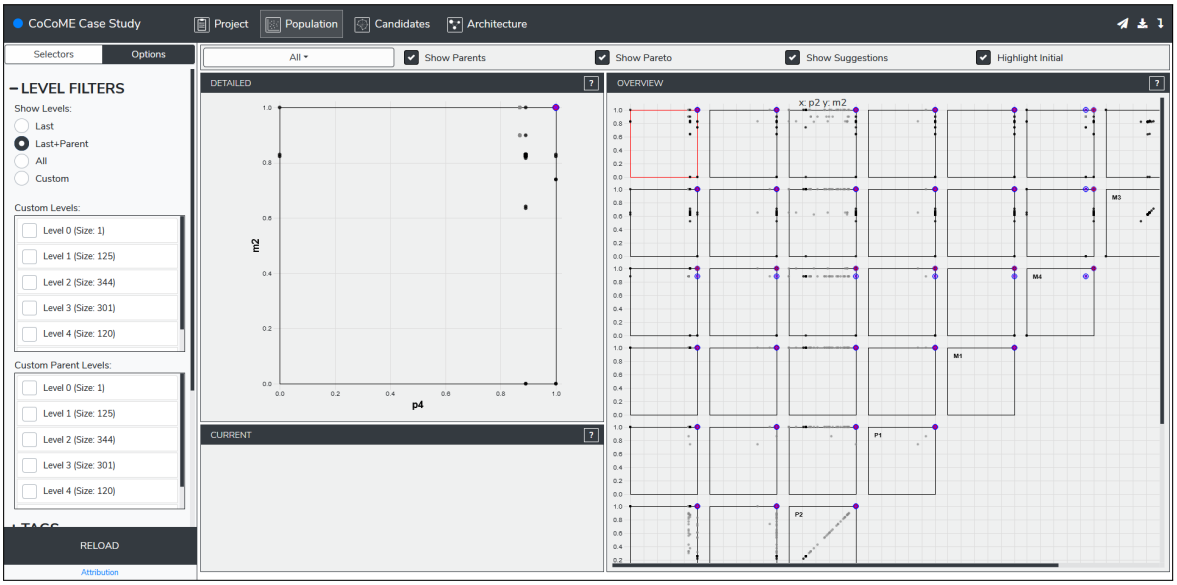

Figure 6.5.: Population View of *SquatVis* showing level 5 of the CoCoME Case Study.

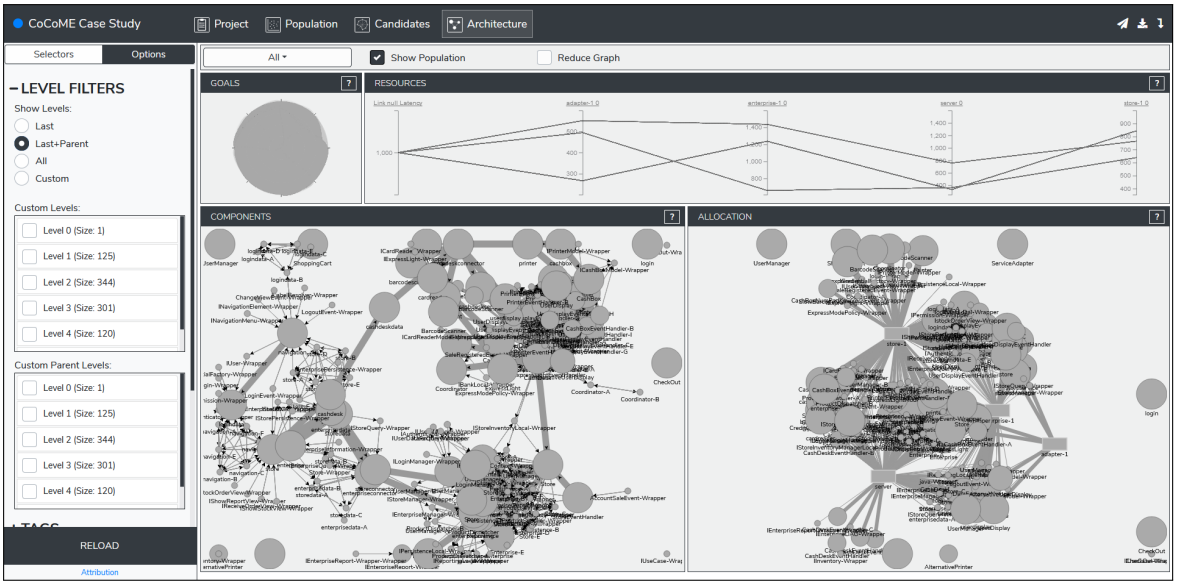

Figure 6.6.: Architecture View of *SquatVis* showing level 5 of the CoCoME Case Study.

6.4. Experiment Settings

A more detailed description of the experiment settings is provided in the following. Section 6.4.1 introduces the experts and the review process. Then, Section 6.4.2 describes the experimental setting of the performance measurements.

6.4.1. Expert Reviews

Three experts have been asked to take part in the expert reviews of *SquatVis*. The experts have been asked to state their expertise in domains related to this work.

Participant A is skilled with regard to the domain of software architecture optimization and the visualization of software systems. Besides, he also is familiar with interactive optimization and PCM. For visualization of multivariate data, he knows the basic concepts. He is highly familiar with the data of the ST+ Case Study and has stated an intermediate familiarity with the CoCoME Case Study. Additionally, he contributed to the development of the state-of-the-art software architecture optimization approaches ArchE [Bac+05; DP+08] and SQuAT [Rag+17]. For these reasons, he can be considered as an expert that can asses this work.

Participant B has high expertise in the domain of software architecture optimization. He is also familiar with visualizations of multivariate data and software systems. Additionally, he knows the basic concepts of interactive optimization and PCM. He is familiar with both case study systems. Furthermore, he also contributed to the development of SQuAT [Rag+17]. Therefore, he also has the expertise to evaluate *SquatVis*.

Participant C has basic knowledge about PCM, the visualization of multivariate data, and software architectures. Besides, he has (almost) no expertise in the domains of software architecture optimization and interactive optimization. While he never heard about the ST+ Case Study, he knows the basics about the CoCoME Case Study. As already two skilled experts from the domain of software architecture optimization are participating, *Participant C* is accepted as a participant for the evaluation despite his limited expertise. The purpose of this decision is also to include the perspective of a layman on *SquatVis*.

The participants received an instruction document that led them through the review process. First, they had to set up *SquatVis* and import the ST+ Case Study and CoCoME Case Study into the tool. A client has been provided, that sends the case study data to *SquatVis*. Therefore, the participants did not have to run SQuAT and all used the same dataset. Next, the participants watched around 20 minutes of video material [Frab]. In these videos, the motivation of the approach, as well as the functionality of the views get explained. Therefore, these videos prepared them to use *SquatVis* on their own.

In the main part of the review, the participants had to solve four tasks for each of the case studies. Each of these tasks is associated with one of the research questions RQ1.1 to RQ1.4. In the first task, the participants had to find 5-10 candidates from the last existing level of the case studies that should be used to generate another level. This task is associated with RQ1.1. In the second task, the participants had to select one of these candidates. They had to explain what changes are necessary to implement this candidate when the initial candidate is the current state of their (imaginary) software system. This task is related to RQ1.3. Then, in the next task, the participants should decide whether the optimization should be continued after the last level. RQ1.2 is associated with this task. Finally, the last task asked them to explain why candidates are good or bad. Therefore, they had to find interrelations between the utility values and the goals or architectures. This task refers to RQ1.4.

The described tasks have been accompanied by a questionnaire that asked the participants to provide feedback. In the first part of the questionnaire, they are asked to state their expertise on relevant research domains, approaches, and the case studies used in this work. Moreover, the participants are asked whether they tried to solve similar problems by visualization in the domain of software architecture optimization and which tools they used to do this.

In the next part of the questionnaire, a set of questions is provided for each of the four tasks. The participants are asked for the time to solve the tasks and whether they actually solved them. In general, the next questions require to specify the results for the tasks and the reasoning behind the choice. As there is not a correct answer, the participants are asked whether they are satisfied with the results. They also need to give reasons for their answers. They are also asked whether *SquatVis* provided enough support, which views, visualizations or features had been essential for solving the task, and which problems they encountered. These questions mostly allow to assess *SquatVis* for RQ1.1 to RQ1.4, but the stated problems also allow to derive improvements as required to answer RQ3.

In the last part of the questionnaire, the questions aim at the overall experience and opinion of the participants. They are asked whether *SquatVis* helped them to foster new insights into the case study, whether it can help to solve tasks like the ones they had to solve in general, or whether it is even superior to the tools they used before to solve such tasks. Concerning RQ2.2, they are also asked to estimate the number of candidates and components that can be visualized by the approach without a degradation of the quality of results. Finally, they are directly asked to suggest improvements and rate them according to their importance. The answers to this question can be used to also answer RQ3.

In general, the questionnaire consists of open questions, to give the participants the freedom to explain their reasoning and experiences. However, in some questions they

are also asked to rate something. For such questions, a Likert scale [Jos+15] with 6 options, ranging from 0 to 5, is used, where 0 is the lowest, and 5 the highest rating. Six answers to do not have a neutral midpoint, which forces the participants to take sides. The exact ratings stated by the participants are also given in the result description in brackets.

The participants had approximately two weeks to set up the prototype and solve the tasks. Apart from that, there have not been any constraints on the invested time or the environment. However, it has been recommended to use a fast computer with at least 4 Gigabytes of memory and a display with a resolution of 1920x1080 pixels.

Besides, *Participant A* and *Participant B* also participated in the pre-evaluation interviews. *Participant A* even attended the initial proposal presentation for the proposed approach. Both received the paper prototype and have been asked to state their opinion about the initial design. They also attended the remote presentation of the first stable version of the prototype and have been asked to suggest improvements.

6.4.2. Performance Measurements

For the time measurements, the CoCoME Case Study and the ST+ Case Study are loaded into *SquatVis*, again. The number of candidates is continuously increased by adding more levels. Thus, the first measurement series is only applied for the initial candidate. In the next measurement series, all candidates from level 1 are added. The candidates from level 2 are then added in the next measurement series, and so on. In the last measurement series, all candidates are active.

In conclusion, six measurement series are conducted for the CoCoME Case Study and three for the ST+ Case Study. Although more measurement series would be desirable, it is acceptable for the intended precision. Nevertheless, only three measurement series for the ST+ Case Study should be considered as borderline, but allow for a rough comparison with the measurements of the CoCoME Case Study.

The measurements are conducted by the developer tools of the Firefox browser [Foub], which show the total loading time of a page. Additionally, browser-side caching has been deactivated. In each measurement series, 10 measurements are conducted by reloading the page by clicking at the reload button of *SquatVis*. To avoid bias, the browser gets restarted if the memory usage exceeds 80%. All measurement series are repeated for the Population View, Candidates View, and the Architecture View. This leads to a total amount of 180 measurements for the CoCoME Case Study and 90 measurements for the ST+ Case Study.

The measurements are conducted on a computer running with Windows 10. The CPU is an Intel Core i7-2600k with four cores and a base clock speed of 3.4 Gigahertz each. The system can use up to 16 Gigabyte of memory. Therefore, it is assumed that the approach is not slowed down by a lack of hardware resources. The views are displayed by Firefox 68.0.2 (64-bit) at a resolution of 1920x1080 pixels in full-screen mode.

6.5. Description of Results

This section describes the results gathered during the evaluation. Section 6.5.1 outlines the outcomes of the pre-evaluation interviews during the development of *SquatVis*. Then, in Section 6.5.2, the results of the expert reviews are described based on the participants' answers to the questionnaire. Finally, the results of the performance measurements to asses the approach's technical scalability are described in Section 6.5.3.

6.5.1. Pre-Evaluation Interviews

During the proposal presentation of this work, *Participant A* was present. With regard to the system's design, he approved that the overall design is reasonable. He also emphasized that an "offline" tool, which stores the received candidates, makes sense to him.

After the paper prototypes have been designed, *Participant A* and *Participant B* were asked to provide feedback. *Participant A* stated that he likes the proposal. *Participant B* also expressed his approval but explicitly asked for the possibility to compare the architectures of two candidates, as this is a typical and time-consuming task for him. He was satisfied when he heard that the Architecture View has a comparison mode. However, he also mentioned that techniques to see more details, especially the internal component behavior, would be appreciated as a future feature.

After the presentation of the early prototypical implementation, *Participant A* and *Participant B* have been asked to provide feedback, again. The feedback was mostly positive. *Participant A* noted that he likes the Architecture View, because it is "interesting" and "experimental", while the Population View and the Candidates View are the rather "classical" views. However, the participants also mentioned potential improvements, which have all been implemented in the final version of *SquatVis*:

- It is sufficient to show the the upper triangle of the matrix in the Population View. Previously the whole matrix was shown.

- The magnified scatter plot in the Population View should also be highlighted in the matrix. This feature was not implemented at this time.
- The initial architecture should be highlighted in the Population View, as it is interesting as a reference. Therefore, it has been proposed to add a tag for the initial candidate.
- The suggested candidates should initially be grouped as *selected*.
- Buttons to group Pareto or suggested candidates as *current* should be added. They are now located below the *all* candidates list.

In addition, the participants stated that the Candidate View is not of high importance in their opinion, due to the existence of the Architecture View. Particular candidates can also be compared to each other in the Architecture View. It has, therefore, been decided to skip its implementation.

6.5.2. Expert Reviews

In the following, the essential answers of the participants are summarized. The summary begins with the previous attempts of the participants in visualizing the results of software architecture optimization. Then, the answers for solving the tasks are outlined in the order of the use cases. The final part of the summary refers to the questions about scalability and suggestions for improvements to the approach.

Previous Experience with Visualizations

Participant A and *Participant B* both tried to visualize the results of software architecture optimization before. The purpose of the visualization was to find “good” candidates (UC1) and free exploration of the results. *Participant A* also tried to find explanations (UC4) for the results, while *Participant B* tried to decide whether the optimization should be continued (UC2). *Participant C* never tried to visualize the results of software architecture optimization, as he did not work in this domain before.

Participant A used various web visualization applications to visualize the response values, architectures, and applied transformations. He also used the Eclipse plugin of ArchE [Bac+05; DP+08] to visualize response values and the Trade Space Visualizer [Stu+04] to get an overview of the design space.

Use Case 1: Candidate Selection

In the first task, the participants had to select 5-10 candidates in level 2 of the ST+ Case Study as an initial population for a new level. *Participant A* solved this task within 4 minutes, while the other participants both required 15 minutes. *Participant C* was the only one who only partially solved the task. Figure 6.7 shows the candidates selected by the participants. *Participant A* only considered Pareto candidates and used the Population View, while the other participants mainly focused on more balanced solutions in the Candidates View.

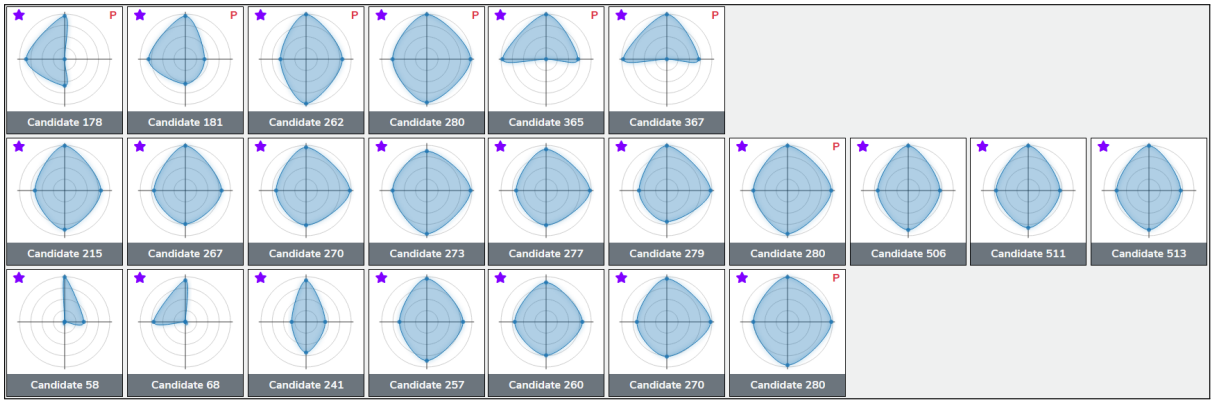

Figure 6.7.: Candidates selected by the participants in the ST+ Case Study. First row: *Participant A*. Second row: *Participant B*. Third row: *Participant C*

Participant A was overall satisfied with the results because they were “easy to select”. *Participant B* was almost fully satisfied and noticed that “some candidates [...] reached a high degree of satisfaction”. *Participant C* was not fully satisfied, because he missed more detailed information about the goals and the right criteria for choosing candidates. *Participant A* (agree: 4) and *participant B* (agree: 5) both agreed that *SquatVis* provided them enough support to solve the task. *Participant C* did not fully agree (agree: 3), for the previously mentioned reasons. *Participant A* missed a way to further zoom into the scatter plots, which influenced his ability to select the right candidates (impact: 3).

For the same task on the CoCoME Case Study, the participants required less time. *Participant A* only invested 3 minutes, while *Participant B* and *Participant C* searched for 10 minutes. However, only *Participant B* stated that he was able to solve the task, while the others solved it partially. Figure 6.8 presents the candidates selected by the participants. *Participant A* and *Participant B* both selected all the three “ideal” candidates. *Participant A* again made his decision based on Pareto optimality, while *Participant B* also considered candidates that have one goal closed to being satisfied. *Participant C* selected its candidates in the Population View and took their improvement into account. *Participant A* and *Participant B* used the Candidates View to solve this task.

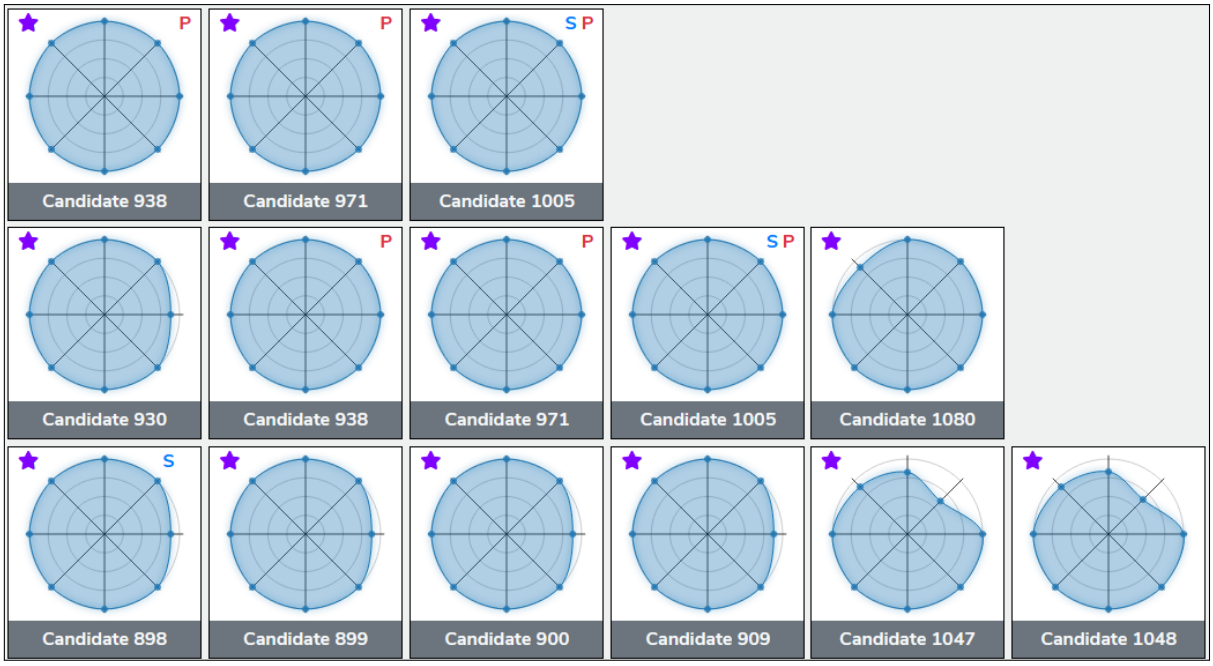

Figure 6.8.: Candidates selected by the participants in the CoCoME Case Study. First row: *Participant A*. Second row: *Participant B*. Third row: *Participant C*

Participant B was the only one who was fully satisfied (5) and felt absolutely (5) supported by *SquatVis*. The other two participants felt only rather satisfied (3). In the opinion of *Participant A*, *SquatVis* provided support (4), while *Participant C* sees more room for improvements (3), but did not have any problems. *Participant A* stated that the missing information about response values had an impact (4) on his user experience.

All the participants agreed (4,5) that *SquatVis* can help architects in selecting candidates (UC1). *Participant A* rather agrees (3) that *SquatVis* can reach better results than the tools he used before.

Use Case 2: Stopping Criterion

In another task, the participants had to decide whether to continue the optimization after level 2 of the ST+ Case Study. *Participant C* decided to skip this task. The other participants were able to solve the task. *Participant A* invested 2 minutes and *Participant B* 10 minutes. Both state that the search should not be continued for the same reason. *Candidate 280* (almost) satisfies three goals and has a high utility for the fourth. Both participants state that this is “good enough”. They mainly used the Candidates View to answer this question and relied on results from the previous tasks.

Both participants are satisfied (4,5) with the result and felt supported (4,5) by *SquatVis*. Besides, they did not experience any problems.

The same decision on the stopping criterion had to be made in the CoCoME Case Study. Again, *Participant C* skipped the task. The other participants solved the task in three minutes or less. Both decided to stop the optimization, as three fully satisfying solutions in terms of utility values have been found. *Participant B* did not state a preference for one of the three solutions. *Participant A* explicitly selected *Candidate 1005* and explained that its "usage of the resource is reasonably low". This shows that he also used the Architecture View to support his decision, while *Participant B* only used the Candidates View.

Like in the same task on the ST+ Case Study, both participants are satisfied (4,5) with the results and felt supported (4,5) by *SquatVis*. Again, they did not experience any problems.

The participants evaluate the ability of *SquatVis* to help in the decision when to stop the optimization (UC2) differently. While *Participant B* fully agrees (5), *Participant A* is less convinced (3). However, both agree (4,5) that *SquatVis* is superior to the tools they used before.

Use Case 3: Candidate Implementation

Concerning the task in which the participants had detected the necessary changes to implement a candidate of the ST+ Case Study, the participants have only been able to partially solve the task. *Participant A* examined *Candidate 178* and worked on this task for 15 minutes. *Participant B* examined *Candidate 280* and worked for 10 minutes. *Participant C* worked for 25 minutes and examined *Candidate 58*.

Participant A tried to put the selected candidate and the initial candidate into the *comparison* group. However, he was unable to identify the (architectural) changes, even in the Architecture View. He finally investigated the differences in the utility values in the Population View and the Candidates View. As a result, he was rather unsatisfied (2) and did rather not feel supported (2) by *SquatVis* in this task.

Participant B recognized that an alternative component and a wrapper have to be implemented. He did not mention changes related to allocation or resources. In conclusion, he felt rather satisfied (3) by the result and the assistance of *SquatVis*, but also stated that he does not fully understand the necessary changes. He experienced two problems: On the one hand, he was unable to read all component names due to visual clutter (impact: 3). On the other hand, he could not see how a component is implemented in detail in an architecture (impact: 5).

Participant C did not describe the actual changes but used the Architecture View to propose own improvements. He noticed that “all the components have access to a redundant database which could lead to a bottleneck and performance issues.” He proposes to use “eventual consistency style and split [the] database to smaller ones”. However, he missed a way to see the impact of his decisions. Therefore, he is only rather satisfied (3), but feels supported (4) by *SquatVis* in solving the task.

Participant A and *Participant B* were unable to solve the same task for the CoCoME Case Study, while *Participant C* was partially able to solve it. *Participant A* tried to find changes for 15 minutes, while *Participant B* invested 5 minutes. *Participant C* worked for 30 minutes to solve the task. *Participant A* and *Participant B* both used the Architecture View for their attempts, but state that there are “too many components”. *Participant B* could not improve the situation, even though he “tried to change the graph settings”. *Participant A* was at least able to state the assumption that there are differences in the allocation graph. Both were not satisfied (1,2) with the results and experienced the support by *SquatVis* as limited (2,3). *Participant C* did not mention any changes explicitly and experienced the same problem of visual clutter in the graph-based diagrams. However, he also mentioned that “using comparison with custom color was very helpful”. Thus, he is rather satisfied (3) with the result and felt rather supported (3) in solving the task.

The participants in general rather disagree (1-3) with the statement that *SquatVis* can be used to determine the necessary changes to implement architectures (UC3). Although *Participant A* did disagree (1) with this statement, he only rather disagrees (2) when it comes to the superiority of *SquatVis* with regard to previously used tools.

Use Case 4: Explain Results

Next, the participants had to find explanations for the results in level 2 of the ST+ Case Study. *Participant A* invested 9 minutes and partially solved the task. *Participant B* was unable to solve the task, even after 20 minutes of investigation. *Participant C* partially solved the task after 30 minutes. All the participants mainly used the Candidates View and the Architecture View for this task.

Participant A stated two assumptions with high certainty (4,5). Firstly, he is sure that most of the “best” candidates use the components *Exporter*, *BusinessTripMgmt*, *TripDB*, and *Booking*. Secondly, most candidates have a low resource usage in *DBServer1*, but a mostly high CPU usage of *Server2* and *Server3*. He was rather satisfied (3) with the results and rather supported (3) by *SquatVis*, but mentioned that he “could not reason much about how certain architecture components would affect the satisfaction of particular quality-attribute scenarios”.

Participant C compared two candidates pairwise. He found out that "comparing candidates 45 with candidate 342 shows that usage of FastReadTripDB improve[s] goal 'p1' and does not affect other goals". In another comparison he detected that the "addition of FastWriteTripDB improve goal 'p2' ". The results make him feel satisfied (4) and he felt supported (4) in the task.

Participant C did not understand why *FastReadTripDB* is not allocated at the database server. However, an explanation for such questions can rather be found at the level of the software architecture optimization approach than in the visualization approach. *Participant B* experienced the same problems related to visual clutter in the graph-based visualizations, as already mentioned before. *Participant A* mentioned that the force-directed layout prevented him from rearranging the nodes (impact: 4).

In the same task for the CoCoME Case Study, *Participant A* and *Participant B* have been unable to solve the task, although *Participant A* tried it for 15 minutes and *Participant B* for 10 minutes. *Participant C* stated one assumption. He assumes that allocating components to the server *enterprise-1* improves the satisfaction of performance goals "without incurring other goals". All participants used the Architecture View for this task. *Participant C* also used the Candidates View.

Participant A and *Participant B* both are unsatisfied (1) with their results, but appreciate the support (2,3) by *SquatVis*. *Participant C* felt satisfied (4) and supported (4) in solving the task. The participants could not identify specific problems that had an impact on their user experience. However, *Participant C* noted that it was hard for him to browse the architecture in the component and allocation graph.

The participants' overall evaluation of the assistance by *SquatVis* in the search for explanations of results (UC4) differs. *Participant A* agrees (4) that it can help, although he only rather agrees (3) when it comes to the superiority of *SquatVis* with regard to previously used tools. The other participants rather disagree (2) with the ability of *SquatVis* to provide assistance in these tasks. *Participant B* also sees *SquatVis* as rather not superior (2) to previously used tools.

Finally, it also has to be noted that all participants agreed (4) or rather agreed (3) that solving the tasks fostered new insights into the two case studies. *Participant A* and *Participant B* also fully agreed (5) that *SquatVis* is an important contribution to the field of software architecture optimization. *Participant C* only rather agreed (3).

Scalability

All participants had to guess the maximum number of candidates that can be visible without a significant quality reduction in solving tasks. The estimation of the participant

varies strongly. *Participant B* estimates a limit of 10 candidates. This is a far smaller size than in all the presented levels. *Participant A* estimates a size of 200 candidates. There are some levels in the case studies of a bigger size. *Participant C* even estimated 2000 candidates, which is more than all the candidates of both case studies together.

The participants have also been asked to estimate the scalability with regard to the number of components. Here, the estimations do not differ that much. *Participant B* estimates 7 components, while *Participant C* estimates 10 components. However, this is less than present in both case studies. *Participant A* suggests a limit of 30 components, which would still include small systems like the ST+ Case Study.

Improvements

Before the participants were asked to suggest improvements, they had the chance to mention what they liked about *SquatVis*. *Participant B* emphasizes that the Candidates View is "very good" and that he liked the concept of grouping candidates. *Participant A* liked the Candidates View as well as the comparison mode. Although "other tools can also give this kind of support", *Participant A* also liked the Comparison View. *Participant C* concluded that he liked the Population View and the possibility to see how candidates evolve.

The critique of the participants mainly focused on the Architecture View. *Participant B* stated that this view "didn't feel [...] intuitive enough". He suggests developing a solution that focuses on two candidates and their differences. *Participant C* emphasizes that a representation of components and allocation focusing on "large scale systems" would be a "very important improvement". *Participant A* also would like (importance: 3) to see improvements in the Architecture View. Additionally, he would like to see features that help "to understand the tactics or design decisions behind a given candidate" (importance: 3). An essential addition (importance: 5) would be the support of response values instead of only utility values.

6.5.3. Performance Measurements

In the following, the results of the time measurements for the Population View, the Candidates View, and the Architecture View are described. Both, the results for the ST+ Case Study and CoCoME Case Study are shown.

Figure 6.9 presents the measured response times for the Population View. A correlation between the number of candidates and the response time can be assumed through visual observation. However, the relation for the *CoCoME Case Study* appears not to be linear.

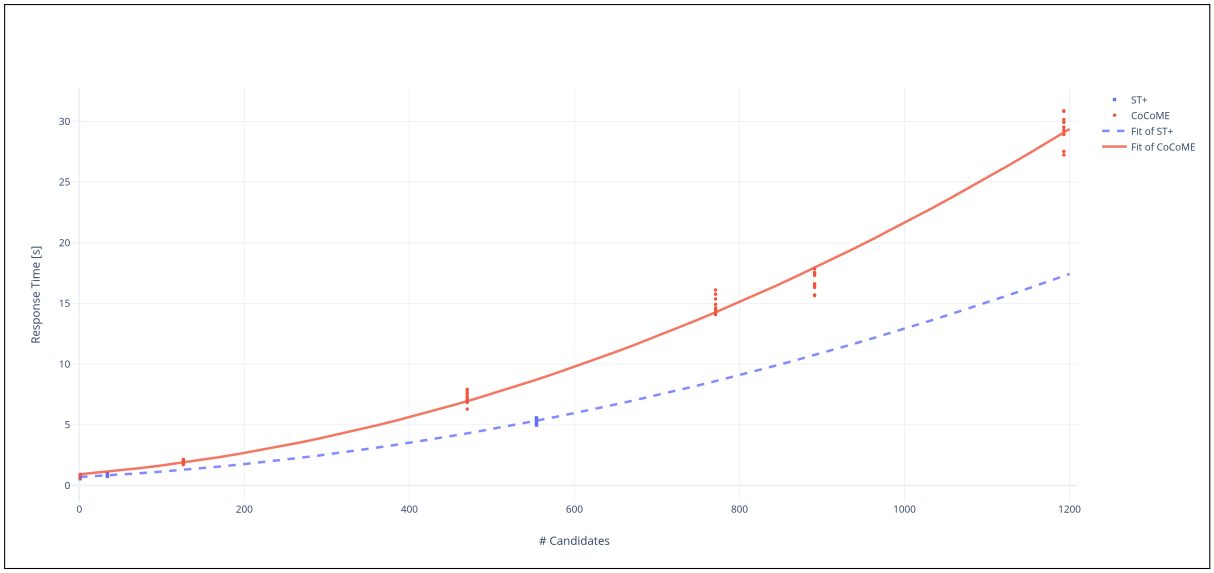

Figure 6.9.: Loading time of the Population View for the case studies ST+ and CoCoME.

Therefore, a quadratic regression is applied, which delivers the following regression formula:

$$f(x) = a \cdot x^2 + b \cdot x + c, a \approx 0.000015, b \approx 0.0058, c \approx 0.92 \tag{6.1}$$

The coefficient of determination $R^2 \approx 0.992$ affirms the assumed relationship. According to this fit, 137 candidates can be displayed within 2 seconds, 609 within 10 seconds, and 1804 within 60 seconds.

The same quadratic regression is applied to the results of the *ST+ Case Study*, which results in the following formula:

$$f(x) = a \cdot x^2 + b \cdot x + c, a \approx 0.0000086, b \approx 0.0036, c \approx 0.69 \tag{6.2}$$

The coefficient of determination $R^2 \approx 0.996$ affirms the assumed relationship. According to this fit, 232 candidates can be displayed within 2 seconds, 850 within 10 seconds, and 2422 within 60 seconds.

The measurements for the Candidates View are displayed in Figure 6.10. Again, a correlation between the number of candidates and the response time seems to exist. It can be analyzed by the use of a quadratic regression for the results of the *CoCoME Case Study*. The following formula is computed by the quadratic regression:

$$f(x) = a \cdot x^2 + b \cdot x + c, a \approx 0.0000085, b \approx 0.0073, c \approx 0.67 \tag{6.3}$$

The coefficient of determination $R^2 \approx 0.995$ affirms the assumed relationship. According to this fit, 154 candidates can be displayed within 2 seconds, 703 within 10 seconds, and 2248 within 60 seconds.

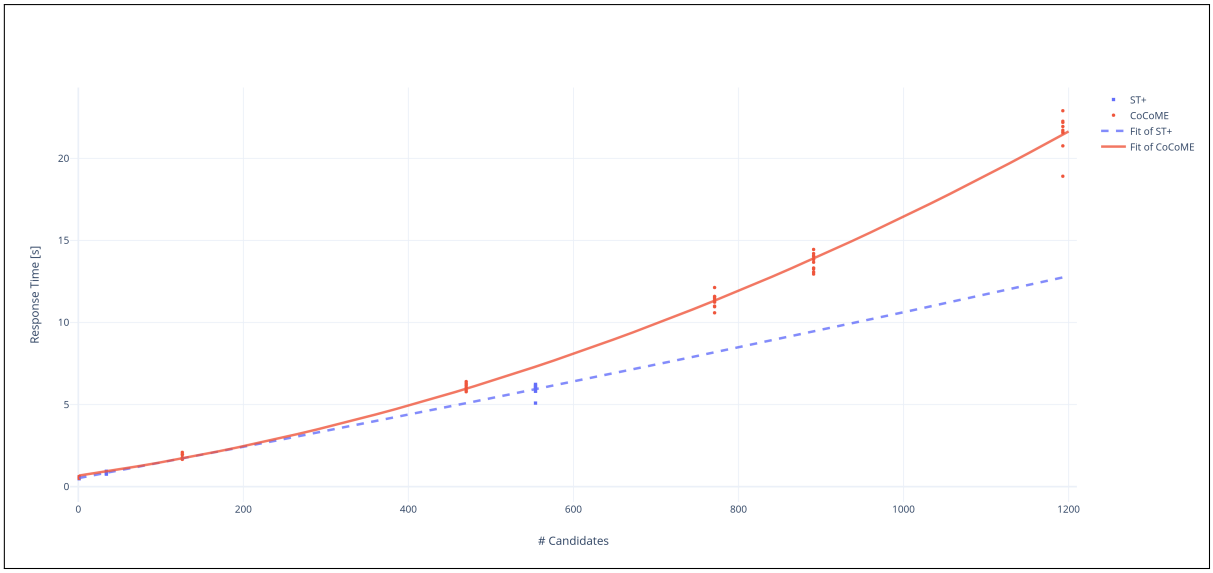

Figure 6.10.: Loading time of the Candidates View for the case studies ST+ and CoCoME.

Applying the quadratic regression to the measurements of the *ST+ Case Study* results in the following formula:

(6.4) $f(x) = a \cdot x^2 + b \cdot x + c, a \approx 0.00000074, b \approx 0.0094, c \approx 0.52$

The coefficient of determination $R^2 \approx 0.995$ affirms the assumed relationship. According to this fit, 156 candidates can be displayed within 2 seconds, 942 within 10 seconds, and 4647 within 60 seconds.

Finally, Figure 6.11 visualizes the measurements for the Architecture View. Although a quadratic relationship is not that obvious, a quadratic regression is applied to the measurements of the *CoCoME Case Study*. The regression delivers the following formula:

(6.5) $f(x) = a \cdot x^2 + b \cdot x + c, a \approx 0.000012, b \approx 0.0046, c \approx 1.57$

The coefficient of determination $R^2 \approx 0.965$ affirms the assumed relationship. According to this fit, 77 candidates can be displayed within 2 seconds, 665 within 10 seconds, and 2014 within 60 seconds.

Quadratic regression for the measurements of the *ST+ Case Study* results in the following formula:

(6.6) $f(x) = a \cdot x^2 + b \cdot x + c, a \approx -0.0000014, b \approx 0.0039, c \approx 0.69$

The coefficient of determination $R^2 \approx 0.980$ affirms the assumed relationship. According to this fit, 395 candidates can be displayed within 2 seconds, while an unlimited number of candidates can be displayed withing 10 and 60 seconds.

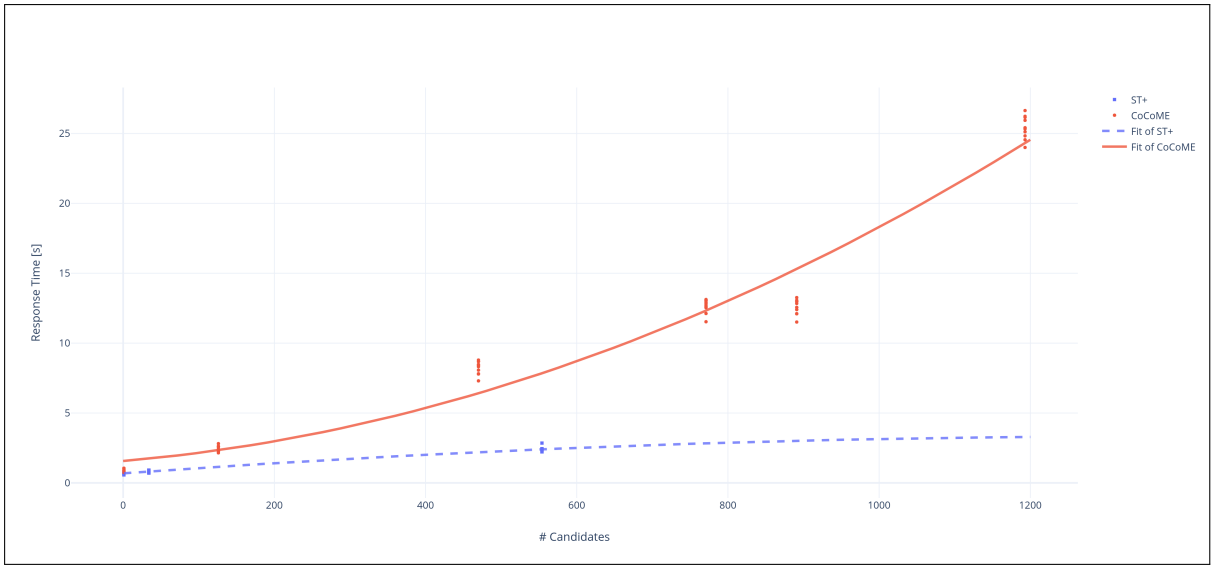

Figure 6.11.: Loading time of the Architecture View for the case studies ST+ and CoCoME.

6.6. Discussion of Results

In this section, the results of the evaluation are discussed and, therefore, the answers to the research questions are given. Section 6.6.1 focuses on the functionality of *SquatVis* with regard to RQ1. Then, Section 6.6.2 discusses the results of the evaluation on scalability as stated in RQ2. Potential improvements, as required by RQ3, are discussed in Section 6.6.3. Finally, Section 6.6.4 identifies potential threats to validity in the performed evaluation.

6.6.1. Functionality

The answers of *Participant C* clearly show that *SquatVis* is designed to be used by experts. Although there are introduction videos and tooltips in the applications, this can not replace knowledge about software architecture optimization, the optimized system, and the specified goals. Therefore, many more actions would have to be taken to make the proposed approach more understandable for laymen. However, software architecture optimization is usually not conducted by laymen, that is why evolution in this direction is of low priority. Additionally, the answers of the domain experts *Participant A* and *Participant B* should be rated as more representative for the potential user community of such an approach.

The previous attempts to visualize the results of software architecture optimization show that the stated use cases describe real problems of researchers in the domain. Only implementing solutions (UC3) was not performed by any of the participants. The reason for this could be that the experts are researchers and not practitioners who optimize existing software systems.

It can be concluded that *SquatVis* can assist architects in selecting candidates for the next level as stated in RQ1.1. The participants unanimously agreed with this statement. Furthermore, they stated that the approach is superior to other ones they used before. Additionally, their results in solving the tasks support this conclusion, because the participants were mostly able to solve the tasks and reported satisfaction with the results and the provided support. This holds for both case studies. Therefore, also the over 500 candidates in level two of the ST+ Case Study and the eight goals in the CoCoME Case Study did not have a big, negative impact. Moreover, two participants found the "ideal" solutions in the CoCoME Case Study.

It is also noteworthy that *Participant A* and *Participant B* selected different candidates and stated different criteria for their selection. While *Participant A* relied on Pareto candidates, *Participant B* aimed for balanced utility values. They even used different views to solve the task. This shows that it is important to consider that experts can have different preferences, which require different views and information.

Concerning RQ1.2, it can also be concluded that *SquatVis* can assist architects in the decision of whether to stop the optimization. This is due to the fact that the two domain experts have been able to solve the corresponding tasks for the ST+ Case Study and the CoCoME Case Study. However, the decisions were comparatively easy for the CoCoME Case Study, as there have been "ideal" solutions. The decisions were more difficult for the ST+ Case Study, in which both experts accepted the same trade-off. However, this also means that the experts have not considered other criteria, e.g., the overall evolution of whole levels. Still, the experts rate the results and the support as satisfying. Additionally, the approach is denoted as superior to previously used tools for this use case.

It has to be concluded that *SquatVis* can not sufficiently assist architects in implementing a selected candidate as stated in RQ1.3. All the related tasks could at most be partially solved. Although a few changes at the level of components have been identified by one expert, the participants did not state specific changes on the resources and the allocation. However, even the found changes are not detailed enough for a concrete implementation in the opinion of the experts.

The participants' answers and results indicate that *SquatVis* offers only limited assistance for exploring and explaining results as stated in RQ1.4. *Participant B* could not even solve the corresponding tasks. However, *Participant A* and *Participant B* were able to

make a few assumptions with high certainty for the smaller software system in the ST+ Case Study. For the CoCoME Case Study, all participants complained about problems with visual clutter and overload, despite the implemented features to mitigate these effects, e.g., small and big component filters. Therefore, the size of the system currently seems to influence the degree of assistance that can be provided.

All in all, the experts approved the approach as an important contribution to the domain of software architecture optimization and also received some new insights into the case studies. Therefore, a positive answer can be given for RQ1, although not all of the subordinate research questions can be answered positively. This is not a perfect solution, but a solid foundation which requires improvements to assist architects in more use cases in the future.

6.6.2. Scalability

With regard to the technical scalability of the proposed approach (RQ2.1), it can be said that there is no simple answer. If fast view switching is required, the number of candidates should be at most between 77 and 395, depending on the model and the view. For a smaller number of view switches, this range can be increased to 609 to 942. It even increases to 1804 to 4647 candidates for minimized view switching. However, there seem to be other variables influencing the loading times of the views than just the number of candidates. The results show significant differences between the two case study systems. In the Architecture View of the ST+ Case Study, the number of candidates had almost no effect. It is reasonable to assume that the number of components in the case study is more important than the number of candidates for this view. The measurements for the Population View also shows a difference between the two case studies. As the number of scatter plots, and therefore visible elements, grows quadratically with the number of goals, this could explain this difference.

However, assuming that the ST+ Case Study is a rather small case study, while the CoCoME Case Study is a rather big one, they can be seen as the best and the worst case in practice. Adding the assumption that view switching is neither used excessively nor reduced to a minimum, the number of candidates is roughly limited to several hundred. Thus, it also seems to be reasonable to add filters for candidates, goals, and components. This is especially important if the data is sent by third parties, which might not consider the described limitations.

Although the range of the ideal number of candidates estimated by the participants was quite big, the results of the tasks itself help to answer the question about scalability with regard to the architect's capabilities (RQ2.2). The number of candidates in level 5 of the CoCoME Case Study is 302 and in level 2 of the ST+ Case Study it is 520.

The participants have been able to solve or partially solve most of the tasks and no participant stated that the number of candidates was a problem. Additionally, there is no other indication that solving the tasks became more difficult in the bigger sized ST+ Case Study. Thus, it can be concluded that handling several hundred candidates is at least possible. For this reason, several hundred candidates seem to be the rough limit of the approach from the technical and architect's point of view.

Considering the participants' estimate on the ideal number of components, filtering for components is at least necessary for large software systems as in CoCoME. However, the participants even experienced visual clutter in the smaller ST+ Case Study and were unable to solve some of the tasks referring to architectural information. Therefore, the approach does not seem to scale well with regard to the size of the model's architecture. However, this is also a problem of node-link diagrams in general.

6.6.3. Improvements

The focus of improvement should lie on the Architecture View, which was the subject of most of the critique by the participants. One option is to modify the existing view. *Participant A* mentioned that it did not allow him to adjust the layout to his liking. Therefore, an option to freeze the force-directed layout simulation could help to solve this problem. Alternatively, a more sophisticated layout algorithm or other settings could be used to reduce visual clutter. Zooming and/or more space for the graphs could help, too.

Another option is to add additional views. For a more detailed description of changes, clicking at a component could open another view, which shows the actual behavior of the component. This could make the approach more useful in identifying changes (UC3) for the implementation of a candidate. It is also possible to design a view that only focuses on the comparison of two candidates as suggested by *Participant B*. Some features of the initially designed Candidate View (see Section 4.6.5) could be adapted.

There is also the option to improve the training of the architects who are using the Architecture View. This view is more complex and experimental than the others but offers also more features and freedom. Therefore, training could improve the architects' performance in handling visual clutter and finding points of interest.

Apart from the improvement of the Architecture View, *Participant A* also suggested other changes. The most important one to him would be to support the visualization of response values. However, visualizing them generically and consistently is more challenging than for utility values. Adding a zoom feature in the Population View would also be useful to *Participant A*.

It is also possible to provide more information to display. As motivated by *Participant C*, a description of the goals could help to understand them even if the architect is not familiar with the specified goals. In general, more features for non-expert architects could be added to increase the number of potential users of *SquatVis*. More information could also be gathered by the interface, e.g., about the used transformations in the software architecture optimization approach.

6.6.4. Threats to Validity

There are some possible threats to the validity of the expert review and the time measurements. These are summarized in the following.

In the expert review, the participants bring in a huge amount of uncertainty. There is the possibility that the chosen experts do not represent the potential user group of *SquatVis* well. They could be more or less skilled than an ordinary expert, which can lead to better or worse results in some tasks. In addition, their interpretations of the questions and the semantics of the options can add bias to the answers, which makes the answers less comparable. The experts' motivation and willingness to invest time also plays an important role, e.g., they could try to finish a task more quickly than a real user would do. Additionally, attitude towards the approach could influence their answers.

Eliminating these human factors is almost impossible, but a bigger group of experts would reduce these threats to validity. However, finding experts who want to participate in a time-consuming review is hard. Additionally, the number of experts is not uncommon for a qualitative evaluation [Ise+13], as stated before. It also has to be considered that the results do not only base on the evaluation of the participants, but also on their performance in solving the tasks.

The method to gather the answers of the participants could have affected the outcome. There is the possibility that the participants did not fully describe their actions in textual form or did not try to resolve uncertainties about the tasks, questions, or the approach itself. Therefore, personal interviews might be better suited for such a qualitative evaluation than a questionnaire. However, the decision to choose a questionnaire has been made due to practical reasons, e.g., the physical distance to the experts. Furthermore, videos have been provided to give a detailed explanation of the approach and its functionality.

The assigned tasks could also have an impact on the results. They may be more or less easy to solve than usual for the investigated use cases. Additionally, the whole case study system could not be a suitable representation of real-world systems. However, the number of available case study systems for SQuAT [Rag+17] is limited and it

is a time-consuming task to design new ones. Furthermore, the participants require knowledge about the case studies to interpret the results. Therefore, the chosen case study systems are the only ones that fulfill the requirements to be used for the evaluation at the moment.

The conducted time measurements to evaluate the scalability of the approach also have some weaknesses. The number of measurement points is too limited to select an appropriate predictive model to describe the loading times of the views in general. The coefficient of determination might also be high for other models, e.g., a linear relationship. This holds especially for the ST+ Case Study with only three series of measurements. Additionally, there can be other effects on the loading time of the views as indicated by the results, e.g., number of goals and components. Moreover, there are also other metrics than loading time of views that are relevant for scalability, e.g., the visualization might not respond even though the page loaded fast.

However, all these decisions have been made for practical reasons. Firstly, a precise prediction is not required. Secondly, the number of candidates can only be controlled through the selection of levels. However, the number of levels is limited by the case studies. Modifying the case studies or using synthetic data could lead to a reduced validity of the results for real systems. Thirdly, a meaningful investigation of other effects, e.g., number of goals and components, would require more case studies, which do not exist. Finally, loading time is a simple metric, which is easy to measure in practice. No other metrics with these properties have been found.

Chapter 7

Conclusion

This work investigated strategies and techniques to assists architects in typical use cases of software architecture optimization by visualization and interaction. Therefore, it interconnected the domains of software architecture optimization, interactive optimization, and information visualization. From the latter one, existing visualization types for the representation of multivariate data have been introduced, e.g., scatter plot matrices, parallel coordinate plots, and radar charts. Additionally, techniques for displaying evolutionary data, e.g., using visualizations with a slider, and for software architectures, which are often represented as graph-based visualizations, have been investigated. A closer examination of state-of-the-art approaches for software architecture optimization showed that their capability to visualize architectural proposals and their properties is limited, while some do not even provide visualizations at all. For this reason, architects have to rely on generic visualization approaches and visual model editors, which are not specifically designed to assist them in their tasks.

Four typical use cases in the domain of software architecture optimization have been identified. Architects need to select candidates based on their preferences and decide whether the search should be continued. Additionally, they need to understand the applied changes to implement an accepted proposal. Furthermore, architects want to reason about the proposed solutions and explain their particular nature. A domain analysis indicated that information about goals, the satisfaction of the goals, as well as the architectures themselves can be imported. It has been decided to use a simplified architectural model and unified utility values to allow for compatibility with most of the existing software architecture optimization approaches and generic result handling. Based on that, techniques to support the architects' tasks have been suggested, e.g., tagging and grouping of proposals. Then, suitable visualizations for different kinds of imported data have been identified, e.g., radar charts to compactly display the satisfaction of goals. These visualizations have been assembled to combine their strengths, which resulted in views.

A prototype consisting of most of these views and techniques has been implemented. Although the prototypical implementation focused on the SQuAT approach and its underlying architectural modeling language PCM, it can also be used by other software architecture optimization approaches. This is due to the generic interface and the modular design. The prototype basically consists of a visualization server for the back-end, while the views are displayed in a browser using web technologies. This even allows to setup the server remotely, which makes it usable in distributed and microservice-based systems.

Based on this prototype, an evaluation has been conducted to assess the functionality and scalability of the approach, as well as to identify potential improvements. A qualitative evaluation setup has been chosen in which domain experts tried to solve tasks on two case study datasets of different size. These tasks referred to the identified use cases. Additionally, the evaluation of the scalability also comprised a quantitative investigation based on time measurements of the views' loading times.

The results of the qualitative evaluation showed that the approach is suitable to assist architects in selecting candidates and deciding whether the optimization should proceed. With regard to finding explanations for the optimization results, only limited support for the smaller case study has been demonstrated. However, the assistance of architects in the implementation of accepted proposals has not been sufficient. The results on scalability indicate that the approach can handle up to several hundred architectural proposals and their properties. However, the number of goals and the size of the architectures also seemed to influence the measurements, so there is not just one factor that influences the number in practice. The answers of the domain experts indicate that improvements should focus on the visualization of the architectural data. In addition, the domain experts recommended additional features, e.g., visualizing response values.

It can be concluded that the approach fulfills its purpose in assisting architects for some of the typical tasks in software architecture optimization. Therefore, the approach is a useful complement to existing software architecture optimization approaches. Nevertheless, this work is only a step in the direction to facilitate and investigate the efficient participation of humans in software architecture optimization. The approach needs to be improved and extended in order to support more use cases and modeling languages. Even more importantly, it needs to be established in the field and must prove its usefulness in practice.

Future Work

As stated before, the proposed approach needs to be improved. This has also been indicated by the domain experts who evaluated the approach. Therefore, future work should first aim at the suggestions by the domain experts. Especially, methods to prevent visual cluttering in the graphs have to be examined. A more sophisticated layout algorithm or the option to rearrange the contents manually could be useful.

It also needs to be investigated how the use case regarding assistance for architects in implementing proposals can be better supported. According to the domain experts, more details about the inner behavior of components have to be shown to achieve this goal. This requires the design of new views and a revision of the architectural analysis to provide additional information. Besides, a new view for detailed, architectural comparison is required if a revised Architecture View still can not achieve the domain experts' approval. The proposed Candidate View could be a useful starting point for this task.

New features would also be appreciated by the domain experts. They suggested to visualize the actual response values instead of utility values, but also more information about the goals and the transformations used in the optimization process. Additionally, some of the made design decisions could be amended, e.g., utility functions could be added to the approach or multiple parents for a generated architectures could be supported. Further investigation is necessary to determine the required changes to implement these features. Then, the new features have to be implemented and it has to be examined whether these changes increase the provided assistance in practice. As with all the other improvements, close collaboration with the domain experts is required.

With regard to the scalability of the approach, the number of displayable proposals turned out to be limited. Conceptually and technically this limit seems to be reached for several hundred candidates, but also goals and components seem to be of importance. Therefore, displaying more proposals would require conceptual as well as technical improvements. It appears to be more promising to focus on methods to filter the existing information than to revise the concepts and the whole implementation. The proposed Goal View, with its option to deactivate goals, and the proposed goal filters, which filter candidates based on utility values of specific goals, could be starting points for future improvements to the scalability of the approach.

Additionally, further evaluation of the approach for the use in practice should be conducted. This requires to integrate the prototype in existing software architecture optimization approaches instead of only working with case study datasets. An integration into SQuAT could be a natural, first step, but other approaches should also be considered in the long term. This would show that the approach can be used by different types

of software architecture optimization approaches. However, also support for other architectural modeling languages than PCM has to be added to achieve this goal.

Finally, the approach also needs to be evaluated by more domain experts. Experts from the domain of interactive optimization and information visualization could help to improve specific aspects of the approach. However, also more experts in software architecture optimization should be involved to be able to consider their specific needs and preferences. As soon as the approach reaches a more mature state, also a quantitative evaluation with more domain experts should be considered.

Appendix A

Expert Review Evaluation Documents

No.	Filename	Description	Length
1	1-Introduction.mp4	This video gives a brief introduction to software architecture optimization, outlines the problems *SquatVis* tries to solve, and explains important terms.	4:04 min
2	2-Projects.mp4	This video introduces the administrative views Projects View and Project View.	2:44 min
3	3-Toolbar.mp4	In this video, the toolbar is introduced. This includes an explanation of groups and global project settings.	4:27 min
4	4-Population.mp4	This video introduces the Population View.	2:42 min
5	5-Candidates.mp4	This video introduces the Candidates View	2:28 min
6	6-Architecture.mp4	This video introduces the Architecture View	4:31 min

Table A.1.: Overview of the *SquatVis* Showcase videos [Frab] presented to the participants of the expert review.

SquatVis Expert Review

Instruction Document

Introduction

This is the **instruction document** for the SquatVis Expert Review. This document will guide you through the steps in the review process. Although this review requires you to go through the steps on your own, you can always ask the organizer of this review for help, whenever necessary.

Name: Sebastian Frank
E-Mail: [redacted]

Steps

1. Please read and sign the **consent form**. You should have received the consent form together with this document.
2. You have to setup the **visualization server** on a local machine. Simply follow the instructions on GitHub. Do not skip the step that increases the memory used by Glassfish. You do NOT need to setup the environment for development.
   Link: https://github.com/SQuAT-Team/squat-vis/wiki/Installation-&-Development
3. Import the **case studies** by running the test clients, which are provided on the release page. Extract the archive and follow the instructions in the README-file. Also read the important notes below after the import terminated.
   Link: https://github.com/SQuAT-Team/squat-vis/releases
4. Watch the six **showcase videos** that are linked on the GitHub page to make yourself familiar with the SquatVis prototype.
   Link: https://www.youtube.com/watch?v=ZZtjrj2e_ms&list=PLJGDPyV4M90uorucX4koZnHMj_eq5pl77
5. Prepare the **questionnaire**. You can either use the printable version that you received together with this document, or follow the link to Google Forms for the digital version. If you want to send a scanned document, please try to write clear and readable. You can add additional pages, if there is not enough space to answer a question. In case you want to use the digital version, note that SquatVis is designed for use in fullscreen mode.
   Link: https://forms.gle/veJNMrTdMbkUDvx7A
6. **Answer** the questions in the questionnaire and solve the described tasks.
7. **Send** your results in an e-mail. You need to attach
   - the signed consent form
   - the scanned questionnaire (if not already submitted via Google Forms)

Important notes:

- After receiving the data, the analysis of the architecture is still running. Thus, the graph view usually takes much longer than the other views to be ready to use. You will see a warning in the graph view, if some architectures are not analyzed yet. I suggest to wait until all architectures are analyzed before really using the tool.
- On my computer, analyzing the ST+ architectures takes less than 5 minutes. CoCoME unfortunately takes 1-2h.
- You can activate *options > level filters > all* and go to the graph view to see how many architectures are already analyzed. Use the reload button to update the view! A browser page refresh (F5) or selecting another view will NOT update the counter. If there is no change within some minutes, then usually something is broken. Then you can i) retry ii) look into the Glassfish server log or iii) send an e-mail and ask for help.
- There is also a third client that loads a test project. This client will print an exception. You can ignore it, if the client still terminates.
- The ST+ dataset does not support 'suggestions'. Thus, there are initially no “selected” candidates.

Thanks for your participation!

University of Stuttgart
Germany

Reliable Software Systems Research Group
(RSS), ISTE

Sebastian Frank

Consent Form

DESCRIPTION: You are invited to participate in **an expert review** on **evaluating the strengths and weaknesses of the SquatVis prototype for typical use cases in the domain of interactive software architecture optimization.**

TIME INVOLVEMENT: Your participation will take approximately **2 - 4 hours.**

DATA COLLECTION: For this study you will use the SquatVis prototype to solve tasks. You will be asked to report your findings and your usage behavior. Also, you will need to fill in questionnaires. Please note that this is an expert review and the number of experts in this domain might be small. Although your name will not visible in publications, there is always the risk that your answers reveal your identity.

RISKS AND BENEFITS: No risk associated with this study. The collected data is securely stored. We do guarantee no data misuse. Please note that this expert review is part of a Master's Thesis. The questionnaire and your answer might have to be fully or partially published as part of the thesis or its supplementary materials. The results of this research study may be presented at scientific or professional meetings, or published in scientific journals. You can decide to deny your consent to the publication, but then your results can not be reported.

PARTICIPANT'S RIGHTS: If you have read this form and have decided to participate in this study, please understand your **participation is voluntary** and you have the **right to withdraw your consent or discontinue participation at any time without penalty or loss of benefits to which you are otherwise entitled. The alternative is not to participate.** You have the right to refuse to answer particular questions. You have the right to suspend participation and continue at a later point in time.

CONTACT INFORMATION: If you have any questions, concerns or complaints about this research, its procedures, risks and benefits, contact following persons:
Sebastian Frank ([redacted])

QUESTIONNAIRE DATA: (select one)

- ☐ Please **do not publish** the questionnaire and my answers.
- ☐ I allow you to **publish** the **anonymous** questionnaire and my answers.

By signing this document I confirm that I agree to the terms and conditions.

Name: _____ *Signature, Date:* _____

SquatVis Expert Review

This questionnaire consists of four parts. In the first part you are asked to provide some personal information. In the second part you have to solve tasks using the SquatVis prototype and the ST+ Case Study. You will be asked to report your results. In the third part you have to solve the same tasks for the CoCoME Case study. The final part is about the SquatVis prototype and your experiences in this review study in general.

Before you continue, you should have successfully installed the visualization server. Assure that also the two before mentioned case studies are already imported and the analysis of the architectures has terminated. Read this expert study's instruction document, if you are not sure. You will also be asked to report the duration that you needed to solve the tasks. Therefore, you should have a device for time measurements at hand. You should also carefully read the questions that you are asked to answer before solving the task. It is suggested to take notes.

Please remember that you can always decide to skip questions if you do not want to answer them. You are allowed to discontinue the case study at any time and return later. The electronic form of this form even allows you to edit your answer. To be sure, try this now, if you plan to use this feature.

*Required

Personal Information

Please fill in your name. It will not be published.

1. **Your Name** *

Please state your level of expertise in the following fields:

2. **Interactive Optimization**
   Mark only one oval.

	0	1	2	3	4	5	
unskilled	◯	◯	◯	◯	◯	◯	very skilled

3. **Software Architecture Optimization**
   Mark only one oval.

	0	1	2	3	4	5	
unskilled	◯	◯	◯	◯	◯	◯	very skilled

4. **Palladio Component Model**
   Mark only one oval.

	0	1	2	3	4	5	
unskilled	◯	◯	◯	◯	◯	◯	very skilled

5. **Visualization of multivariate data**
   Mark only one oval.

	0	1	2	3	4	5	
unskilled	◯	◯	◯	◯	◯	◯	very skilled

6. **Visualization of software architectures**
   Mark only one oval.

	0	1	2	3	4	5	
unskilled	◯	◯	◯	◯	◯	◯	very skilled

Please state your familiarity with the following case studies:

7. **Simple Tactics Plus (ST+)**
   This is a rather small model of a Business Trip Management system. The Simple Tactics model is provided as part of the architecture optimization approach 'PerOpteryx' as an example system. This model has been extended to Simple Tactics Plus. It is presented in the research paper 'Distributed Quality-Attribute Optimization of Software Architectures'. To model is available at https://github.com/SQuAT-Team/paper-supplementary/tree/master/SimpleTacticsPlus.
   Mark only one oval.

	0	1	2	3	4	5	
unknown	◯	◯	◯	◯	◯	◯	very familiar

8. **Common Community Modeling Example (CoCoME)**
   As the name indicates, this model has been designed for the use as a common example system of large scale. It is also modeled as instance of the Palladio Component Model. For the use in SQuAT, some adjustments have been made (extensions, added alternative components, removed composite components, ...). The initial implementation of the model can be found at https://github.com/cocome-community-case-study. The version used in SQuAT and the review is available at https://github.com/SQuAT-Team/kamp-test/tree/master/squat-tool/models/cocomeWithResourceDemands.
   Mark only one oval.

	0	1	2	3	4	5	
unknown	◯	◯	◯	◯	◯	◯	very familiar

9. **Please list projects related to the before mentioned fields and case studies you participated in. Also mention the field or case study.**

10. **Did you (try to) visualize the results of software architecture optimization before?**
Mark only one oval.

- ◯ Yes
- ◯ No

11. **Please list the tools/approaches that you used to visualize the results. Also mention whether you used it to visualize response values, architectures, both, or something else.**
Example: PCM Eclipse Plugin - Architecture

_____

_____

_____

_____

_____

12. **What have been the goals of visualizing the results?**
Tick all that apply.

- ☐ Find "good" candidates
- ☐ Decide whether a next iteration is required
- ☐ Determine changes in a single candidate
- ☐ Find explanations
- ☐ Free exploration
- ☐ Other: _____

ST+ Case Study

In this section you have to solve four tasks using the SquatVis prototype and the dataset of the ST+ Case Study. Please open the according project now. Assure that the project is in the initial state. Remember to read the questions before solving the tasks. Please note that this case study does not support 'suggested' candidates!

TASK I: Select Candidates

In this task you have to find candidates that should be used as initial population to generate the next level of search. Select candidates which are in your oppinion the most suitable ones for this purpose. You have to find at least five candidates, but not more than ten. The selected candidates have to be part of level 2 of the ST+ Case Study. You are allowed to have a look at previous levels if you want to, but you are not allowed to select them.

13. **Have you been able to solve the task?**
Mark only one oval.

- ◯ Yes
- ◯ Partially
- ◯ No

14. **How much time did it take to solve the task?**

_____

15. **Which candidates did you select?**
Please provide the IDs of these candidates.

16. **Why did you select these candidates?**

17. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	6	7	8	9	10	
unsatisfied	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	very satisfied

18. **Why are you (un)satisfied with the results?**

19. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

20. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

21. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK II: Implementing solutions

Let's assume that the initial candidate represents an existing software system. Your task is to determine the changes that are necessary to transform the initial candidate into another solution. Choose one of your selected candidates from Task I. Note that you do not really have to implement the system, you only have to determine the changes that would be necessary to do it.

22. **Have you been able to solve the task?**
Mark only one oval.

- Yes
- Partially
- No

23. **How much time did it take to solve the task?**

24. **Which candidate did you select?**
Please provide the name of the candidate.

25. **List the necessary changes.**

26. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied							very satisfied

27. **Why are you (un)satisfied with the results?**

28. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

29. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

30. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK III: Explainability

Your task is to explore a whole level and make assumptions about why candidates satisfy or do not satisfy the specified goals. Also look for interesting patterns and relationships between goals in general, utility values, and architectures. Again, only take level 2 of the ST+ Case Study into consideration.
Examples:
i) Most candidates that satisfy a specific goal use the same alternative component.
II) Linear dependency between two goals.

31. **Have you been able to solve the task?**
Mark only one oval.

- ◯ Yes
- ◯ Partially
- ◯ No

32. **How much time did it take to solve the task?**

33. **List your assumptions. For each assumption, also rate how certain (0-5 scale, 0: uncertain, 5: certain) you are that this assumption is correct.**
Example: Usage of component 'FastDB' improves goal 'm1' - Certainty: 4

34. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

35. **Why are you (un)satisfied with the results?**

36. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

37. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

38. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK IV: Stop Search

Have a look at the whole dataset of the ST+ Case Study again. Decide whether the search should be continued or can be terminated after level 2. Try to find as many arguments as possible by exploring the dataset to support your decision.
Note:
A reason to terminate the search is usually either a satisfying solution or the insight that a satisfying solution can not be reached.

39. **Have you been able to solve the task?**
Mark only one oval.

- Yes
- Partially
- No

40. **How much time did it take to solve the task?**

41. **Which candidate did you select?**
Please provide the name of the candidate.

42. **Should the search be continued?**
Mark only one oval.

- Yes
- No
- I don't know

43. **List the reasons that support your decision.**

44. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

45. **Why are you (un)satisfied with the results?**

46. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

47. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

48. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

CoCoME Case Study

In this section you have to solve four tasks using the SquatVis prototype and the dataset of the CoCoME Case Study. Please open the according project now. Assure that the project is in the initial state and all architectures are analyzed. Remember to read the questions before solving the tasks.

TASK I: Select Candidates

In this task you have to find candidates that should be used as population to generate the next level of search. Select candidates which are in your oppinion the most suitable ones for this purpose. You have to find at least five candidates, but not more than ten. The selected candidates have to be part of level 5 of the CoCoME Case Study. You are also allowed to have a look at previous levels if you

want to, but you are not allowed to select them.

49. **Have you been able to solve the task?**
Mark only one oval.

- ◯ Yes
- ◯ Partially
- ◯ No

50. **How much time did it take to solve the task?**

_____

51. **Which candidates did you select?**
Please provide the IDs of these candidates.

_____

_____

_____

_____

_____

52. **Why did you select these candidates?**

_____

_____

_____

_____

_____

53. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

54. **Why are you (un)satisfied with the results?**

_____

_____

_____

_____

_____

55. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

56. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

57. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK II: Implementing solutions

Let's assume that the initial candidate represents an existing software system. Your task is to determine the changes which are necessary to to transform the initial candidate into another solution. Choose one of your selected candidates from Task I. Note that you do not really have to implement the system, you only have to determine the changes that would be necessary to do it.

58. **Have you been able to solve the task?**
Mark only one oval.

- ◯ Yes
- ◯ Partially
- ◯ No

59. **How much time did it take to solve the task?**

60. **Which candidate did you select?**
Please provide the name of the candidate.

61. **List the necessary changes.**

62. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

63. **Why are you (un)satisfied with the results?**

64. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

65. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

66. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK III: Explainability

Your task is to explore a whole level and make assumptions about why candidates satisfy or do not satisfy the specified goals. Also look for interesting patterns and relationships between goals in general, utility values, and architectures. Again, only take level 5 of the CoCoME Case Study into consideration.
Examples:
i) Most candidates that satisfy a specific goal use the same alternative component.
II) Linear dependency between two goals.

67. **Have you been able to solve the task?**
Mark only one oval.

- ◯ Yes
- ◯ Partially
- ◯ No

68. **How much time did it take to solve the task?**

_____

69. **List your assumptions. For each assumption, also rate how certain (0-5 scale, 0: uncertain, 5: certain) you are that this assumption is correct.**
Example: Usage of component 'FastDB' improves goal 'm1' - Certainty: 4

_____

_____

_____

_____

_____

70. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

71. **Why are you (un)satisfied with the results?**

_____

_____

_____

_____

_____

72. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

73. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

_____

_____

_____

_____

_____

74. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

TASK IV: Stop Search

Have a look at the whole dataset of the CoCoME Case Study again. Decide whether the search should be continued or can be terminated after level 5. Try to find as many arguments as possible by exploring the dataset to support your decision.
Note:
A reason to terminate the search is usually either a satisfying solution or the insight that a satisfying solution can not be reached.

75. **Have you been able to solve the task?**
Mark only one oval.

- Yes
- Partially
- No

76. **How much time did it take to solve the task?**

77. **Which candidate did you select?**
Please provide the name of the candidate.

78. **Should the search be continued?**
Mark only one oval.

- Yes
- No
- I don't know

79. **List the reasons that support your decision.**

80. **Are you satisfied with the results?**
Mark only one oval.

	0	1	2	3	4	5	
unsatisfied	◯	◯	◯	◯	◯	◯	very satisfied

81. **Why are you (un)satisfied with the results?**

82. **Did the SquatVis prototype provide enough support to solve this task?**
Mark only one oval.

	0	1	2	3	4	5	
absolutely not	◯	◯	◯	◯	◯	◯	absolutely

83. **Which views/visualizations/features of SquatVis have been essential for solving the task.**

84. **List the problems (bugs, bad performance, missing features, ...) that you encountered while solving the task. Also rate the impact (0-5 scale, 0: no impact, 5: huge impact) of these problems on your user experience.**
Example: Candidates View loading time > 10s - Impact: 4

Follow-up Questions

This section asks some general questions about your experience with the SquatVis prototype.

In the following you have to decide whether you rather agree or disagree with presented statements.

85. **Solving the tasks fostered new insights into the used case studies.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

86. **SquatVis can help architects in selecting candidates for the next level.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

87. **SquatVis is superior to other tools that I used before for this task.**
This statement refers to the previous one. 'Superior' means results have better quality or are reached faster/easier resulting in similar quality.
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

88. **SquatVis can help architects to determine the changes necessary to implement architectures.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

89. **SquatVis is superior to other tools that I used before for this task.**
This statement refers to the previous one. 'Superior' means results have better quality or are reached faster/easier resulting in similar quality.
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

90. **SquatVis can help architects in finding explanations for their results.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

91. **SquatVis is superior to other tools that I used before for this task.**
This statement refers to the previous one. 'Superior' means results have better quality or are reached faster/easier resulting in similar quality.
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

92. **SquatVis can help architects to decide when to stop the optimization.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

93. **SquatVis is superior to other tools that I used before for this task.**
This statement refers to the previous one. 'Superior' means results have better quality or are reached faster/easier resulting in similar quality.
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

94. **The SquatVis approach in general is an important contribution to the field of Software Architecture Optimization.**
Mark only one oval.

	0	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	◯	agree

Scalability

You are asked to evaluate the conceptual scalability of the SquatVis prototype based on your experiences in solving the tasks. Neglect the effect of loading times.

95. **Guess the maximum number of candidates that can be visible without a significant quality reduction in solving tasks.**

96. **Guess the maximum number of components that can be visible without a significant quality reduction in solving tasks.**

Improvements

By answering these questions you can help to improve the SquatVis prototype.

97. **What did you like about the SquatVis prototype?**

98. **What did you dislike about the SquatVis prototype?**

99. **What can be done to improve the SquatVis prototype? Which views/visualizations /features should be added? Please also weight your suggestions (0-5 scale, 0: unimportant, 5: very important)?**

Thanks for your participation!

Appendix

Bibliography

[Ahn+11] J.-w. Ahn, M. Taieb-Maimon, A. Sopan, C. Plaisant, B. Shneiderman. "Temporal visualization of social network dynamics: Prototypes for nation of neighbors." In: *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*. Springer. 2011, pp. 309–316 (cit. on p. 17).

[Aig+11] W. Aigner, S. Miksch, H. Schumann, C. Tominski. *Visualization of time-oriented data*. Springer Science & Business Media, 2011 (cit. on p. 17).

[Ale+09] A. Aleti, S. Bjornander, L. Grunske, I. Meedeniya. "ArcheOpterix: An extendable tool for architecture optimization of AADL models." In: *Model-based Methodologies for Pervasive and Embedded Software*. IEEE. 2009, pp. 61–71 (cit. on pp. 11, 20, 26, 27).

[Ale+13] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya. "Software architecture optimization methods: A systematic literature review." In: *IEEE Transactions on Software Engineering* 39.5 (2013), pp. 658–683 (cit. on pp. 2, 8–11, 29, 35, 36).

[And72] D. F. Andrews. "Plots of high-dimensional data." In: *Biometrics* (1972), pp. 125–136 (cit. on p. 15).

[BBW83] J. Bertin, W. J. Berg, H. Wainer. *Semiology of graphics: diagrams, networks, maps*. Vol. 1. 0. University of Wisconsin press Madison, 1983 (cit. on p. 13).

[BC+06] E. Bondarev, M. Chaudron, et al. "A process for resolving performance trade-offs in component-based architectures." In: *International Symposium on Component-Based Software Engineering*. Springer. 2006, pp. 254–269 (cit. on pp. 9, 27).

[BC87] R. A. Becker, W. S. Cleveland. "Brushing scatterplots." In: *Technometrics* 29.2 (1987), pp. 127–142 (cit. on p. 14).

[BC99] H. Biermann, R. Cole. “Comic strips for algorithm visualization.” In: *New York University, New York, NY* (1999) (cit. on p. 17).

[BI96] B. Boehm, H. In. “Identifying quality-requirement conflicts.” In: *IEEE software* 13.2 (1996), pp. 25–35 (cit. on p. 1).

[BKR07] S. Becker, H. Koziolek, R. Reussner. “Model-based performance prediction with the Palladio Component Model.” In: *Proceedings of the 6th international workshop on Software and Performance*. ACM. 2007, pp. 54–65 (cit. on pp. 27, 61, 65).

[BOH11] M. Bostock, V. Ogievetsky, J. Heer. “D3 Data-Driven Documents.” In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (Dec. 2011), pp. 2301–2309 (cit. on p. 65).

[Bac+05] F. Bachmann, L. Bass, M. Klein, C. Shelton. “Designing software architectures to achieve quality attribute requirements.” In: *IEEE Software* 152.4 (2005) (cit. on pp. 12, 19, 20, 26–28, 90, 94).

[Bos] M. Bostock. *D3.js*. URL: https://d3js.org (cit. on p. 65).

[Bou+19] S. Bougouffa, K. Busch, R. Heinrich, A. v. Hoorn, M. Konersmann, S. Seifermann, E. Taşpolatoğlu, F. Ocker, C. Vargas, M. Fahimipirehgalin, R. Reussner, B. Vogel-Heuser. “Case Studies for the Community.” In: *Managed Software Evolution*. Ed. by R. Reussner, M. Goedicke, W. Hasselbring, B. Vogel-Heuser, J. Keim, L. Märtin. Springer International Publishing, 2019, pp. 335–374. ISBN: 978-3-030-13499-0. DOI: 10.1007/978-3-030-13499-0_12. URL: https://doi.org/10.1007/978-3-030-13499-0_12 (cit. on p. 86).

[Bra+08] J. Branke, J. Branke, K. Deb, K. Miettinen, R. Slowiński. *Multiobjective optimization: Interactive and evolutionary approaches*. Vol. 5252. Springer Science & Business Media, 2008 (cit. on pp. 2, 21).

[Bru09] J. Brutlag. *Speed matters for Google web search*. 2009 (cit. on p. 1).

[CKK02] P. Clements, R. Kazman, M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. SEI series in software engineering. Addison-Wesley, 2002 (cit. on pp. 3, 8, 27, 83, 86).

[CLV06] C. A. C. Coello, G. B. Lamont, D. A. V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006 (cit. on p. 11).

[CM84] W. S. Cleveland, R. McGill. “The many faces of a scatterplot.” In: *Journal of the American Statistical Association* 79.388 (1984), pp. 807–822 (cit. on pp. 14, 42).

[Car99] M. Card. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999 (cit. on p. 17).

[Ces+03] A. Cesta, G. Cortellessa, A. Oddi, N. Policella. "A CSP-based interactive decision aid for space mission planning." In: *Congress of the Italian Association for Artificial Intelligence*. Springer. 2003, pp. 511–522 (cit. on p. 21).

[Che73] H. Chernoff. "The use of faces to represent points in k-dimensional space graphically." In: *Journal of the American Statistical Association* 68.342 (1973), pp. 361–368 (cit. on p. 15).

[DKL09] S. Duszynski, J. Knodel, M. Lindvall. "SAVE: Software architecture visualization and evaluation." In: *European Conference on Software Maintenance and Reengineering*. IEEE. 2009, pp. 323–324 (cit. on p. 23).

[DLR09] G. M. Draper, Y. Livnat, R. F. Riesenfeld. "A survey of radial methods for information visualization." In: *IEEE transactions on visualization and computer graphics* 15.5 (2009), pp. 759–776 (cit. on p. 41).

[DMW08] T. Dwyer, K. Marriott, M. Wybrow. "Dunnart: A constraint-based network diagram authoring tool." In: *International Symposium on Graph Drawing*. Springer. 2008, pp. 420–431 (cit. on p. 21).

[DP+08] J. A. Diaz-Pace, H. Kim, L. Bass, P. Bianco, F. Bachmann. "Integrating quality-attribute reasoning frameworks in the ArchE design assistant." In: *International Conference on the Quality of Software Architectures*. Springer. 2008, pp. 171–188 (cit. on pp. 12, 19, 20, 26–28, 90, 94).

[DPC08] J. A. Diaz-Pace, M. Campo. "Exploring Alternative Software Architecture Designs: A Planning Perspective." In: *IEEE Intelligent Systems* 23.5 (2008) (cit. on pp. 12, 20, 26).

[Deb54] G. Debreu. "Representation of a preference ordering by a numerical function." In: *Decision processes* 3 (1954), pp. 159–165 (cit. on p. 34).

[Dix+97] A. Dix, J. Finley, G. Abowd, R. Beale. "Human-computer interaction." In: (1997) (cit. on p. 17).

[Duc14] J. Duckett. *Web Design with HTML, CSS, JavaScript and jQuery Set*. Wiley Publishing, 2014 (cit. on p. 65).

[ETB11] U. Erdemir, U. Tekin, F. Buzluca. "E-Quality: A graph based object oriented software quality visualization tool." In: *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. IEEE. 2011, pp. 1–8 (cit. on p. 23).

[Ede02] A. H. Eden. “LePUS: A visual formalism for object-oriented architectures.” In: *The 6th World Conference on Integrated Design and Process Technology*. Citeseer. 2002, pp. 26–30 (cit. on p. 23).

[Eve78] B. S. Everitt. *Graphical techniques for multivariate data*. North-Holland, 1978 (cit. on p. 15).

[FG12] P. H. Feiler, D. P. Gluch. *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*. Addison-Wesley, 2012 (cit. on pp. 9, 27, 35).

[FR91] T. M. Fruchterman, E. M. Reingold. “Graph drawing by force-directed placement.” In: *Software: Practice and experience* 21.11 (1991), pp. 1129–1164 (cit. on pp. 16, 45).

[Fei04] P. Feiler. “Open Source AADL Tool Environment (OSATE).” In: (Jan. 2004) (cit. on p. 22).

[Fek09] J.-D. Fekete. “Visualizing networks using adjacency matrices: Progresses and challenges.” In: *2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics*. IEEE. 2009, pp. 636–638 (cit. on p. 16).

[Foua] E. Foundation. *Eclipse GlassFish*. URL: https://eclipse-ee4j.github.io/glassfish (cit. on p. 65).

[Foub] M. Foundation. *Firefox*. URL: https://www.mozilla.org (cit. on p. 92).

[Fraa] S. Frank. *SquatVis Prototype*. URL: https://github.com/SQuAT-Team/squat-vis (cit. on pp. 5, 61).

[Frab] S. Frank. *Techniques for Visualization and Interaction in Software Architecture Optimization Supplementary Material*. DOI: 10.5281/zenodo.3454747. URL: https://doi.org/10.5281/zenodo.3454747 (cit. on pp. 5, 61, 83, 86, 90, 113).

[Fra16] S. Frank. *Handling Quality Trade-Offs in Architecture-based Performance Optimization*. Bachelor’s Thesis, University of Stuttgart. 2016 (cit. on pp. 82, 83).

[GB08] S. Greenberg, B. Buxton. “Usability evaluation considered harmful (some of the time).” In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2008, pp. 111–120 (cit. on p. 80).

[GHM08] K. Gallagher, A. Hatch, M. Munro. “Software architecture visualization: An evaluation framework and its application.” In: *IEEE Transactions on Software Engineering* 34.2 (2008), pp. 260–270 (cit. on p. 23).

[GK06] E. R. Gansner, Y. Koren. “Improved circular layouts.” In: *International Symposium on Graph Drawing*. Springer. 2006, pp. 386–398 (cit. on p. 16).

[Gar+05] J. J. Garrett et al. “Ajax: A new approach to web applications.” In: (2005) (cit. on p. 65).

[Gie+03] H. Giese, S. Burmester, F. Klein, D. Schilling, M. Tichy. “Multi-agent system design for safety-critical self-optimizing mechatronic systems with UML.” In: *OOPSLA*. 2003, pp. 21–32 (cit. on p. 9).

[Gil92] A. C. Gillies. *Software quality: theory and management*. Chapman & Hall, Ltd., 1992 (cit. on pp. 1, 8).

[Gla01] R. L. Glass. “Frequently forgotten fundamental facts about software engineering.” In: *IEEE Software* 3 (2001), pp. 112–110 (cit. on p. 1).

[Goo+17] S. Goodwin, C. Mears, T. Dwyer, M. G. de la Banda, G. Tack, M. Wallace. “What do constraint programming users want to see? Exploring the role of visualisation in profiling of models and search.” In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 281–290 (cit. on p. 24).

[HLD02] H. Hauser, F. Ledermann, H. Doleisch. “Angular brushing of extended parallel coordinates.” In: *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*. IEEE. 2002, pp. 127–130 (cit. on p. 14).

[HR01] E. Hart, P. Ross. “GAVEL - a new tool for genetic algorithm visualization.” In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 335–348 (cit. on p. 20).

[Hol06] D. Holten. “Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data.” In: *IEEE Transactions on visualization and computer graphics* 12.5 (2006), pp. 741–748 (cit. on p. 16).

[Ins85] A. Inselberg. “The plane with parallel coordinates.” In: *The visual computer* 1.2 (1985), pp. 69–91 (cit. on p. 14).

[Ise+13] T. Isenberg, P. Isenberg, J. Chen, M. Sedlmair, T. Möller. “A systematic review on the practice of evaluating visualization.” In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2818–2827 (cit. on pp. 80, 107).

[Iso] *Systems and software engineering – Systems and software Quality Requirement and Evaluation (SQuaRE)– System and software quality models*. Standard. Geneva, CH: International Organization for Standardization, 2011 (cit. on pp. 39, 62, 63).

[Jai90] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990 (cit. on p. 8).

[Jar89] S. L. Jarvenpaa. “The effect of task demands and graphical format on information processing strategies.” In: *Management science* 35.3 (1989), pp. 285–303 (cit. on p. 14).

[Joh+83] C. John, W. Cleveland, B. Kleiner, P. Tukey. “Graphical methods for data analysis.” In: *Pacific Grove: Wadsworth* (1983), pp. 158–62 (cit. on p. 14).

[Jon94] C. V. Jones. “Visualization and optimization.” In: *ORSA Journal on Computing* 6.3 (1994), pp. 221–257 (cit. on p. 21).

[Jos+15] A. Joshi, S. Kale, S. Chandel, D. Pal. “Likert scale: Explored and explained.” In: *British Journal of Applied Science & Technology* 7.4 (2015), p. 396 (cit. on pp. 13, 92).

[KC00] H.-S. Kim, S.-B. Cho. “Application of interactive genetic algorithm to fashion design.” In: *Engineering applications of artificial intelligence* 13.6 (2000), pp. 635–644 (cit. on p. 21).

[KH81] B. Kleiner, J. A. Hartigan. “Representing points in many dimensions by trees and castles.” In: *Journal of the American Statistical Association* 76.374 (1981), pp. 260–269 (cit. on p. 15).

[KL83] J. B. Kruskal, J. M. Landwehr. “Icicle plots: Better displays for hierarchical clustering.” In: *The American Statistician* 37.2 (1983), pp. 162–168 (cit. on pp. 16, 59).

[KR08] H. Koziolek, R. Reussner. “A model transformation from the Palladio Component Model to Layered Queueing Networks.” In: *Performance Evaluation: Metrics, Models and Benchmarks*. Springer, 2008, pp. 58–78 (cit. on p. 10).

[KR11] A. Koziolek, R. Reussner. “Towards a generic quality optimisation framework for component-based system models.” In: *14th International ACM Sigsoft Symposium on Component Based Software Engineering CBSE*. 2011, pp. 103–108 (cit. on pp. 11, 20, 26–28, 32, 83).

[KS76] J. G. Kemeny, J. L. Snell. *Markov Chains*. Springer-Verlag, New York, 1976 (cit. on p. 9).

[Keh+12] T. Kehrer, U. Kelter, M. Ohrndorf, T. Sollbach. “Understanding model evolution through semantically lifting model differences with SiLift.” In: *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. IEEE. 2012, pp. 638–641 (cit. on p. 23).

[Kla+02] G. W. Klau, N. Lesh, J. Marks, M. Mitzenmacher. “Human-guided tabu search.” In: *AAAI/IAAI*. 2002, pp. 41–47 (cit. on p. 21).

[Kor91] P. Korhonen. “Using harmonious houses for visual pairwise comparison of multiple criteria alternatives.” In: *Decision Support Systems* 7.1 (1991), pp. 47–54 (cit. on p. 15).

[Koz14] A. Koziolek. *Automated improvement of software architecture models for performance and other quality attributes*. Vol. 7. KIT Scientific Publishing, 2014 (cit. on pp. 9, 11).

[LL07] M. Lungu, M. Lanza. "Exploring inter-module relationships in evolving software systems." In: *11th European Conference on Software Maintenance and Reengineering, 2007. (CSMR'07)*. IEEE. 2007, pp. 91–102 (cit. on p. 23).

[LS04] J. D. Lee, K. A. See. "Trust in automation: Designing for appropriate reliance." In: *Human factors* 46.1 (2004), pp. 50–80 (cit. on pp. 3, 29, 33).

[LSP05] G. Langelier, H. Sahraoui, P. Poulin. "Visualization-based Analysis of Quality for Large-scale Software Systems." In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*. ASE '05. Long Beach, CA, USA: ACM, 2005, pp. 214–223 (cit. on p. 24).

[Li+11] R. Li, R. Etemaadi, M. Emmerich, M. Chaudron. "An evolutionary multiobjective optimization approach to component-based software architecture design." In: *IEEE Congress on Evolutionary Computation*. 2011, pp. 432–439 (cit. on pp. 11, 19, 26, 27).

[Lun+11] C. Lundstrom, T. Rydell, C. Forsell, A. Persson, A. Ynnerman. "Multi-touch table system for medical visualization: Application to orthopedic surgery planning." In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), pp. 1775–1784 (cit. on p. 81).

[MH11] P. Merkle, J. Henss. "EventSim–an event-driven Palladio software architecture simulator." In: *Palladio Days* (2011), pp. 15–22 (cit. on p. 10).

[MM00] K. Miettinen, M. M. Mäkelä. "Interactive multiobjective optimization system WWW-NIMBUS on the internet." In: *Computers & Operations Research* 27.7-8 (2000), pp. 709–723 (cit. on pp. 21, 22).

[MV15] T. Metsalu, J. Vilo. "ClustVis: a web tool for visualizing clustering of multivariate data using Principal Component Analysis and heatmap." In: *Nucleic acids research* 43.W1 (2015), W566–W570 (cit. on p. 20).

[Mac86] J. Mackinlay. "Automating the design of graphical presentations of relational information." In: *ACM Transactions On Graphics (Tog)* 5.2 (1986), pp. 110–141 (cit. on pp. 13, 38, 40).

[McC76] T. J. McCabe. "A complexity measure." In: *IEEE Transactions on software Engineering* 4 (1976), pp. 308–320 (cit. on pp. 8, 82).

[Mei+15] D. Meignan, S. Knust, J.-M. Frayret, G. Pesant, N. Gaud. "A review and taxonomy of interactive optimization methods in operations research." In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5.3 (2015), p. 17 (cit. on p. 21).

[Mie14] K. Miettinen. “Survey of methods to visualize alternatives in multiple criteria decision making problems.” In: *OR Spectrum* 36.1 (2014), pp. 3–37 (cit. on pp. 14, 22).

[Mou11] R. E. Moustafa. “Parallel coordinate and parallel coordinate density plots.” In: *Wiley Interdisciplinary Reviews: Computational Statistics* 3.2 (2011), pp. 134–148 (cit. on p. 41).

[Mun78] J. B. Munson. “Software maintainability: a practical concern for life-cycle costs.” In: *The IEEE Computer Society’s Second International Computer Software and Applications Conference, 1978. COMPSAC’78.* IEEE. 1978, pp. 54–59 (cit. on p. 1).

[NZES05] P. Ngatchou, A. Zarei, A El-Sharkawi. “Pareto multi objective optimization.” In: *Proceedings of the 13th International Conference on Intelligent Systems Application to Power Systems*. IEEE. 2005, pp. 84–91 (cit. on pp. 9, 28, 31).

[Nah04] F. F.-H. Nah. “A study on tolerable waiting time: how long are Web users willing to wait?” In: *Behaviour & Information Technology* 23.3 (2004), pp. 153–163 (cit. on pp. 1, 81).

[Oraa] Oracle. *Java Server Faces*. URL: http://www.javaserverfaces.org (cit. on p. 65).

[Orab] Oracle. *Java*. URL: https://www.java.com (cit. on p. 65).

[PG88] R. M. Pickett, G. G. Grinstein. “Iconographic displays for visualizing multidimensional data.” In: *Proceedings of the 1988 IEEE Conference on Systems, Man, and Cybernetics*. Vol. 514. 1988, p. 519 (cit. on p. 15).

[PRF17] F. Pfähler, O. Röhrdranz, S. Frank. *A Parallel and Distributed Architecture for Improving the Performance of the SQuAT Optimization Framework*. Guided Research Project, University of Stuttgart. https://goo.gl/eddtUY. 2017 (cit. on pp. 63, 83).

[Pos81] J. Postel. “Transmission control protocol.” In: (1981) (cit. on p. 65).

[Pri] PrimeTek. *Primefaces*. URL: https://www.primefaces.org (cit. on p. 65).

[Pur+08] H. C. Purchase, N. Andrienko, T. J. Jankun-Kelly, M. Ward. “Theoretical foundations of information visualization.” In: *Information Visualization*. Springer, 2008, pp. 46–64 (cit. on p. 3).

[RBJ17] J. Rumbaugh, G. Booch, I. Jacobson. *The unified modeling language reference manual*. Addison Wesley, 2017 (cit. on pp. 9, 22, 35).

[RLN07] R. Rosenholtz, Y. Li, L. Nakano. “Measuring visual clutter.” In: *Journal of vision* 7.2 (2007), pp. 17–17 (cit. on p. 16).

[Rag+17] A. Rago, S. Vidal, J. A. Diaz-Pace, S. Frank, A. van Hoorn. “Distributed quality-attribute optimization of software architectures.” In: *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. ACM. 2017, p. 7 (cit. on pp. 2, 4, 12, 20, 26–28, 61, 64, 65, 82–84, 86–88, 90, 107).

[Rau+08] A. Rausch, R. H. Reussner, R. Mirandola, F. Plasil. *The common component modeling example: comparing software component models*. Vol. 5153. Springer, 2008 (cit. on p. 86).

[Reu+11] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, M. Kuperberg. *The Palladio Component Model*. Tech. rep. KIT, Fakultät für Informatik, 2011 (cit. on pp. 2, 4, 9, 22, 35, 39).

[Rob07] J. C. Roberts. “State of the art: Coordinated & multiple views in exploratory visualization.” In: *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007)*. IEEE. 2007, pp. 61–71 (cit. on p. 18).

[SAA16] H. B. Salameh, A. Ahmad, A. Aljammal. “Software evolution visualization techniques and methods-a systematic review.” In: *7th International Conference on Computer Science and Information Technology (CSIT), 2016*. IEEE. 2016, pp. 1–6 (cit. on p. 23).

[STa] SQuAT-Team. *Common Community Modeling Example Case Study SQuAT Models*. URL: https://github.com/SQuAT-Team/kamp-test/tree/master/squat-tool/models/cocomeWithResourceDemands (cit. on p. 86).

[STb] SQuAT-Team. *Simple Tactics Plus Case Study Models*. URL: https://github.com/SQuAT-Team/paper-supplementary/tree/master/SimpleTacticsPlus (cit. on p. 83).

[SW03] C. U. Smith, L. G. Williams. “Software performance engineering.” In: *UML for Real*. Springer, 2003, pp. 343–365 (cit. on p. 1).

[Sak+14] B. Saket, P. Simonetto, S. Kobourov, K. Börner. “Node, node-link, and node-link-group diagrams: An evaluation.” In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 2231–2240 (cit. on p. 15).

[San95] G. Sander. “A fast heuristic for hierarchical Manhattan layout.” In: *International Symposium on Graph Drawing*. Springer. 1995, pp. 447–458 (cit. on p. 16).

[Shn96] B. Shneiderman. “The eyes have it: A task by data type taxonomy for information visualizations.” In: *Proceedings 1996 IEEE symposium on visual languages*. IEEE. 1996, pp. 336–343 (cit. on p. 17).

[Sny03] C. Snyder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, 2003 (cit. on pp. 30, 80).

[Sof] O. Software. *ObjectDB*. URL: https://www.objectdb.com (cit. on p. 65).

[Ste+08] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008 (cit. on p. 23).

[Ste+46] S. S. Stevens et al. "On the theory of scales of measurement." In: (1946) (cit. on p. 13).

[Str+17] D. Strüber, K. Born, K. D. Gill, R. Groner, T. Kehrer, M. Ohrndorf, M. Tichy. "Henshin: A usability-focused framework for emf model transformation development." In: *International Conference on Graph Transformation*. Springer. 2017, pp. 196–208 (cit. on p. 23).

[Stu+04] G. Stump, M. Yukish, J. Martin, T. Simpson. "The ARL trade space visualizer: An engineering decision-making tool." In: *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. 2004, p. 4568 (cit. on pp. 3, 15, 20, 94).

[TA15] C. Tominski, W. Aigner. *The TimeViz Browser–A Visual Survey of Visualization Techniques for Time-Oriented Data*. 2015 (cit. on p. 17).

[TF98] Y. S. Tan, N. M. Fraser. "The modified star graph and the petal diagram: Two new visual aids for discrete alternative multicriteria decision making." In: *Journal of Multi-Criteria Decision Analysis* 7.1 (1998), pp. 20–33 (cit. on p. 15).

[TI] B. A. Twitter Inc. *Bootstrap*. URL: https://getbootstrap.com (cit. on p. 65).

[TT81] P. A. Tukey, J. W. Tukey. "Preparation; prechosen sequences of views." In: *Interpreting multivariate data* (1981), pp. 189–213 (cit. on p. 13).

[Tec] K. I. of Technology. *Common Community Modeling Example Case Study Original Models*. URL: https://github.com/cocome-community-case-study (cit. on p. 86).

[VBRK92] A. van Vliet, C. G. E. Boender, A. H. G. Rinnooy Kan. "Interactive optimization of bulk sugar deliveries." In: *Interfaces* 22.3 (1992), pp. 4–14 (cit. on p. 21).

[VVL00] D. A. Van Veldhuizen, G. B. Lamont. "Multiobjective evolutionary algorithms: Analyzing the state-of-the-art." In: *Evolutionary computation* 8.2 (2000), pp. 125–147 (cit. on p. 2).

[VWN03] J. J. Van Wijk, W. A. Nuij. "Smooth and efficient zooming and panning." In: *IEEE Symposium on Information Visualization 2003 (IEEE Cat. No. 03TH8714)*. IEEE. 2003, pp. 15–23 (cit. on p. 18).

[WAG06] L. Wilkinson, A. Anand, R. Grossman. “High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions.” In: *IEEE Transactions on Visualization and Computer Graphics* 12.6 (2006), pp. 1363–1372 (cit. on p. 14).

[WBWK00] M. Q. Wang Baldonado, A. Woodruff, A. Kuchinsky. “Guidelines for using multiple views in information visualization.” In: *Proceedings of the working conference on Advanced visual interfaces*. ACM. 2000, pp. 110–119 (cit. on p. 18).

[WEG87] S. Wold, K. Esbensen, P. Geladi. “Principal component analysis.” In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52 (cit. on pp. 15, 20).

[War02] M. O. Ward. “A taxonomy of glyph placement strategies for multidimensional data visualization.” In: *Information Visualization* 1.3-4 (2002), pp. 194–210 (cit. on p. 14).

[War12] C. Ware. *Information visualization: perception for design*. Elsevier, 2012 (cit. on pp. 13, 14).

[War94] M. O. Ward. “Xmdvtool: Integrating multiple methods for visualizing multivariate data.” In: *Proceedings of the Conference on Visualization’94*. IEEE Computer Society Press. 1994, pp. 326–333 (cit. on p. 20).

[Wer23] M. Wertheimer. “Laws of organization in perceptual forms.” In: *A source book of Gestalt Psychology* (1923) (cit. on p. 13).

[Wes+96] D. B. West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ, 1996 (cit. on p. 15).

[Xu+19] K. Xu, Y. Wang, L. Yang, Y. Wang, B. Qiao, S. Qin, Y. Xu, H. Zhang, H. Qu. “CloudDet: Interactive Visual Analysis of Anomalous Performances in Cloud Computing Systems.” In: *IEEE transactions on visualization and computer graphics* (2019) (cit. on p. 80).

[YC88] S. S. Yau, P.-S. Chang. “A metric of modifiability for software maintenance.” In: *Proceedings of the Conference on Software Maintenance, 1988*. IEEE. 1988, pp. 374–381 (cit. on p. 8).

[Yas+19] R. Yasaweerasinghelage, M. Staples, H.-Y. Paik, I. Weber. “Optimising Architectures for Performance, Cost, and Security.” In: *European Conference on Software Architecture*. Springer, 2019, pp. 161–177 (cit. on p. 12).

All links were last followed on September 23, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature