Visualising Software Dynamics through Architecture Mining

Master Thesis Business Informatics

Carlijn Quik
6151000

Department of Information and Computing Sciences
Utrecht University, Princetonplein 5, 3584 CC Utrecht, Netherlands
c.j.m.quik@students.uu.nl
26-09-2019

Primary Supervisor
Dr. J. Hage
J.Hage@uu.nl

External Supervisor
Lucas Jellema
lucas.jellema@amis.nl

Secondary Supervisor
Dr. ir. J.M.E.M. van der Werf
J.M.E.M.vanderWerf@uu.nl

Universiteit Utrecht

Abstract

Techniques that aim at reconstructing the design or architecture of a system, such as software architecture reconstruction, mainly focus on functional aspects of a system and tend to ignore quality attributes [1]. Approaches that do take quality attributes into account emphasise the reconstruction of architectural artefacts, rather than compliance checking of quality attributes during the operational phase of the software [2]. This is why run-time software execution data is most suitable to evaluate and evolve software architecture.

When reconstructing software architecture based on data extracted from a running system, the amount of extracted data can become overwhelming (i.e. dozens of gigabytes) and difficult to comprehend. The field of visual analytics builds better and more effective ways to understand and analyse such large datasets [3]. Despite this, there is a lack of research that studies how dynamic system data can be used to visualise architectural information effectively. Moreover, the gap between research and practice in the field of software architecture visualisation is emphasised by several researchers [4, 5, 6].

We present Architecture Miner, a web-based interactive dashboard that makes it possible to extract architectural intelligence from large-scale Java applications. It provides an overview of both run-time structure and metrics given a scenario chosen by the user. The hierarchical interactions between class objects in Java systems are represented using a node-link diagram that is enhanced with the Group-In-A-Box Layout for Multi-faceted Analysis [7]. It groups nodes that represent class objects into clusters based on their (design-time) package depth. The treemap space filling technique is used to size boxes that contain nodes according to the size of the corresponding cluster. The chronological order of calls is shown using a bar chart with their start time on the x-axes, and their duration on the y-axes.

We applied Architecture Miner to the acceptance environment of a Java application with 225,000 LOC, 1,921 classes and 348 packages. The application is being developed in the Netherlands by a DevOps team consisting of four DevOps Engineers, a Product Owner and a Software Architect. To depict which architectural information would be of value to them, we interviewed the DevOps Engineer with the most experience with the system, the Architect and the Product Owner. After prioritising the requirements through a survey with twenty participants, including the complete DevOps team, five questions were formulated that the dashboard should provide an answer to.

The required data is collected using AJPOLog to instrument the system (AJPOLog stands for AspectJ Partially Ordered Logging tool). AJPOLog captures software behaviour by logging method calls including their caller and callee, and the (design-time) locations of the classes' package from which the caller and callee class objects are instantiated [8]. This enables us to partially extract the path that method calls take through the system, as well as the corresponding method execution times and static location.

It is not possible to capture the functional features and quality attributes of a complex system in a single view that is understandable by, and of value to, all of its stakeholders [9]. For different viewpoints and perspectives in software architecture, different visualisations, formalisms, and abstractions are required. To scope this research, we have focused on architectural information that is valuable to the stakeholders of the system analysed in our case study.

Based on the validation results we can conclude that, within the context of the case study, the designed visualisations are effective in providing valuable architectural information to the system's stakeholders. We consider our results to be a good indication that the insights extracted with Architecture Miner can be valuable within multiple use cases in different contexts.

Keywords— Software Architecture Visualisation, Trace Visualisation, Software Architecture Reconstruction, Group-In-A-Box Layout, Network Analysis

Acknowledgements

First of all, I would like to thank Jurriaan Hage for always being willing to help out and supporting me in the decisions I have made. Next, I would like to thank Jan Martijn van der Werf for his everlasting and inspiring enthusiasm for the topic and his solid advise, in particular his advise to take the Information Visualisation course at Técnico Lisboa – which turned out to be a key source of knowledge to conduct this research.

Special thanks goes out to Lucas Jellema for our many challenging and fruitful discussions and for providing me with a platform to connect with and learn from experts in the field. It has been of tremendous value. Last but not least, I would like to thank the members of the DevOps team: Jeffrey Resodikromo, Rosanna Denis, Michael van Gastel, Cees-Pieter Franx, Marcos Klaver, and Nico Klasens. Their willingness to help out and debate each visualisation and its value has been essential to successfully apply the research in practice.

Contents

1 Introduction

Since a system's software architecture captures its most significant properties and design constraints, changes and additions made to a system that violate its architectural principles can degrade its performance and shorten its useful lifetime [10]. However, the rapidly changing stakeholder requirements and business conditions of a system, and with that the potential frequency and scale of software adaptations, make it difficult to control such architecture erosion. As this happens, architectural documentation often does not stay up to date (if present in the first place) and the people working on the system change over time [11]. At the same time, the architecture of a system cannot be re-established using solely the source code of a system, since it does not explicitly specify it [12].

Van der Werf and Brinkkemper (2017) [1] argue that techniques that aim at reconstructing the design or architecture of a system, such as software architecture reconstruction (SAR), mainly focus on functional aspects of a system and tend to ignore quality attributes. Approaches that do take quality attributes into account emphasise the reconstruction of architectural artefacts, rather than compliance checking of quality attributes during the operational phase of the software [2]. This is why run-time software execution data (SED) is most suitable to evaluate and evolve software architecture. SED are a combination of static context data and behaviour data, and can be collected by applying static and behaviour data extractors to a running system.

The amount of extracted information can, however, become overwhelming (i.e. dozens of gigabytes) and difficult to comprehend. The field of visual analytics builds better and more effective ways to understand and analyse such large datasets [3]. The human brain processes images 60,000 times faster than text, and 90 percent of information transmitted to the brain is visual [13]. The purpose of this research is to explore how visualising the combination of static and dynamic system data can provide valuable architectural information to large-scale system's stakeholders. The main research question therefore is:
RQ: Which visualisation(s) of large-scale system's software dynamics are effective in providing valuable architectural information to its stakeholders?

1.1 Problem statement

When reconstructing software architecture based on data extracted from a running system, the amount of extracted data can become overwhelming (i.e. dozens of gigabytes) and difficult to comprehend. The field of visual analytics builds better and more effective ways to understand and analyse such large datasets [3]. Despite this, there is a lack of research that studies how the combination of static and dynamic system data can be used to visualise architectural information effectively. Moreover, the gap between research and practice in the software architecture visualisation (SAV) field is emphasised by several authors [4, 5, 6]. There is a need to not only consider design principles but also human factors to ensure that techniques and tools satisfy the audience's needs [4].

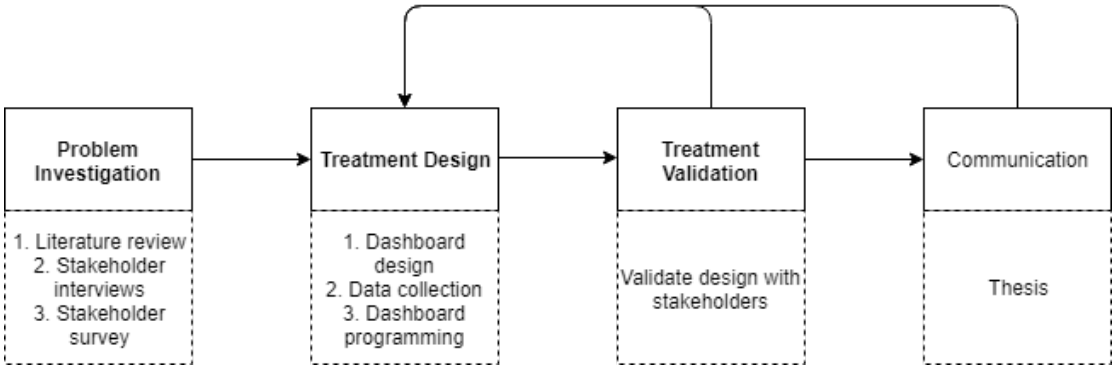
1.2 Research contribution

This research contributes to the field of software architecture by visualising both static and dynamic data of a real-world large-scale software system. Moreover, an application of SAV in practice will be performed, while taking into account stakeholder requirements. Fundamental to this research is the combination of best practices of the fields of software architecture and visual analytics, and additions to the knowledge bases of both.

2 Research Approach

The methods used for this research are structured according to the design science framework by Roel Wieringa [14]. An overview of the research approach is shown in Figure 1. In this chapter, the selection of research methods as shown in this figure is explained in section 2.1. In order to provide an answer to the main research question, several supporting research questions are defined which are set forth per research phase in sections 2.2, 2.3, and 2.4.

Figure 1: *Research approach.*



2.1 Research method selection

To begin with, a *literature review* is conducted to gain understanding of the current state-of-the-art in the several fields that are relevant for this research. In addition, since the gap between research and practice in the field of software architecture visualisation (SAV) is emphasised by several authors [4, 5, 6], *interviews* are conducted with the stakeholders of the system to which the visualisations will be applied in the treatment validation phase. Though this data has poor external validity [15], it will give some insight into how SAV is currently used at the company, how the envisioned visualisations can be valuable to the system's stakeholders, and a means to validate the treatment design. Both the interviewees and the interview questions are chosen based on the literature review. The interview protocol is provided in Appendix A. The interviews are held in the Dutch language and therefore the transcripts are too. The quotes that are used in this report are translated to English and can be found in Appendices A.1, A.2, and A.3.

Based on the interviews, questions are identified that the stakeholders have about the system. Subsequently, these questions are prioritised by the system's stakeholders through a *survey*. The chosen prioritisation technique is ranking, which is based on an ordinal scale, but the requirements are ranked without ties in rank [16]. This means that the most important requirement/question is ranked as 1 and the least important as n (for n requirements, 10 in this case). To see the relative difference between the requirements, and simultaneously align the views of the different subjects, the average priority of each requirement will be calculated. This technique is chosen because it ensures that the stakeholders choose which requirement is more important than another instead of putting equal emphasis on all of them. Compared to a technique in which the stakeholders can distribute a certain amount of points over the requirements, the results are also more representative of the group of subjects as a whole, because one subject putting a requirement in first place which others put in tenth place will not skew the results much. Besides sending the survey to the stakeholders of the system the visualisations will be applied to, the survey is also sent to other IT experts within the same company. The aim of this is to provide a broader perspective on how the visualisations can provide valuable insights to system stakeholders. The survey can be found in Appendix A.

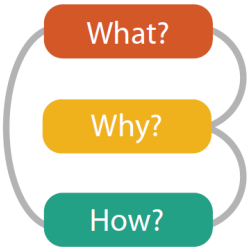
2.2 Problem investigation

The problem investigation phase is meant to explore the domain of the research project and to learn more about the problem that the researchers aim to solve. The part of the report that describes this phase is structured using two frameworks. The first used is Maletic's framework, which presents a space of possible visualisation systems with respect to software engineering tasks [17]. This framework consists of five dimensions which are enumerated below along with the corresponding chapter(s) in this report:

- Target - What data source should be represented? (chapter 3)
- Representation - How should it be represented? (chapter 4)
- Medium - Where should the visualisation be represented? (chapter 4)
- Tasks - Why is the visualisation needed? (chapter 5 and 6 and 7)
- Audience - Who will use the visualisation? (chapter 5 and 6 and 7)

The second framework is the three-part analysis framework for a visualisation instance: why is the task being performed, what data is shown in the views, and how is the visualisation idiom constructed in terms of design choices [18]. This is shown in the figure below and the reason why the "tasks" and "audience", and "representation" and "medium" parts of Maletic's framework are combined into one sub-research question each. These sub-research questions are explained next.

Figure 2: *Three-part analysis framework for a visualisation instance [18].*



SQ 1. How can architectural information be extracted from a running software system?
It is evident that before data can be visualised, it needs to be collected. Ideally, the visualisations should be applicable to various systems in different domains. It is important to gain an understanding of the state-of-the-art in architecture mining to select the right data and corresponding collection methods. This sub-research question will be answered in chapter 3, while simultaneously answering the question within Maletic's framework: "What is the data source to represent?".

SQ 2. Which questions should visualisations of software dynamics be able to answer in order to be valuable to system stakeholders?
Software architecture plays an important role in numerous aspects of software development, such as understanding, reuse, and management [19]. Therefore, there is an abundance of tasks that visualisations can (potentially) support. It is important to identify the tasks that this research should focus on. Since the tasks that can be supported by SAV depend highly on the audience of the visualisations, chapter 5.1 describes the possible stakeholders of SAV and their needs as found in the literature, and chapter 6 summarises the most important results of the interviews with stakeholders of the system under study. In this way, both the "why?" and "who?" questions as defined by Maletic should be answered.

SQ 3. How are visualisation techniques and supporting tools currently used to represent software architecture?
Answering this research question will give insight into the current state-of-the-art in SAV and corresponding tools. Additionally, best practices in the field of information visualisation will be discussed in order to provide a reference against which the quality of the treatment design can be evaluated. This analysis can be found in Chapter 4, which will answer the “how?" and “where?" questions as prescribed by Maletic.

2.3 Treatment design

Based on the results of the problem investigation phase, a dashboard will be designed and programmed. To do this, the most suitable visualisation techniques will be selected, and decisions will be made about which data structures the visualisations will be able to support, how the visualisations are connected to one another, and how the user can interact with them. In this phase, two sub-research questions are answered which are explained next.

SQ 4. Which visualisations can answer the questions identified in SQ 2?
Visualisation design can be broken down into what-why-how questions that have data-task-idiom answers [18]. It is therefore expected that one visualisation will not be sufficient to support all the tasks that are needed to answer the questions that will be identified while answering SQ 2. This research question will help select the right existing visualisation techniques, and reveal whether the design of new ones is needed. While designing the visualisations, the dashboard design and how the user interacts with the visualisations in it should also be taken into account.

2.4 Treatment validation

To verify whether the dashboard provides valuable insights to the stakeholders of the system it is based on, a *case study* will be conducted. The visualisations will be applied to a ‘real-world’ Java application. The stakeholders of the real-world system will be interviewed again to assess whether the dashboard aids in answering the research questions, whether they perceive it as easy to use and useful. Additionally, a focus group will be held with all interviewees to discuss the most important results. This should answer the following sub-research question:

SQ 5. What are the strengths and weaknesses of the created dashboard?
The expected outcome of this is knowledge regarding the value of the dashboard to the stakeholders of the system it will be applied to. Additionally, the improvements that should be made to the visualisations in the future can be identified.

3 Architecture Mining

This section aims at answering the question:

SQ 1. How can architectural information be extracted from a running software system?

It starts with an explanation of software architecture and how it can be represented (section 3.1). After that, the different sources of information that can be used as an input to reconstruct software architecture are set forth (3.2). Next, the architecture mining framework and the use of SED are explained (3.3). The last section contains an analysis of existing tools that can be used to collect SED (3.4).

3.1 Grasping software architecture

According to Garlan [19], software architecture plays an important role in at least the following six aspects of software development: understanding, reuse, construction, evolution, analysis and management. Knowledge about the architecture of the system is therefore vital to fulfill its purpose. Software architecture is defined as follows:

Definition 1 (Software architecture)**.** *Software architecture is the set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both [9].*

Rozanski and Woods (2011) [9] identify two types of structures: static and dynamic. Static structures define a system's internal design-time elements and their arrangement, while dynamic structures define the system's run-time elements and their interactions.

3.1.1 Architectural views

To portray the elements of the architecture that are relevant, architectural views can be used as presented below in definition 2. To construct a view, a viewpoint can be used which is described by definition 3.

Definition 2 (View)**.** *A view is a representation of a set of system elements and relations associated with them [9].*

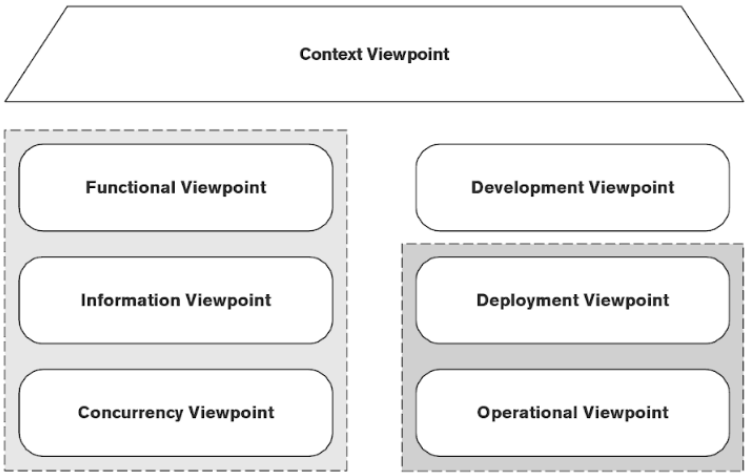
Definition 3 (Viewpoint)**.** *A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views [9].*

The principle behind this is that it is not possible to capture the functional features and quality attributes of a complex system in a single model that is comprehensible for, and of value to, its stakeholders. Therefore, it is more effective to represent a system using a set of interrelated views.

The seven core viewpoints for information systems architecture are shown in Figure 3 [9]. The context viewpoint over-arches the other viewpoints and describes the relationships, dependencies, and interactions between the system and its environment. The environment in this context comprises the people, systems, and other external entities with which the system interacts. The functional, information, and concurrency viewpoints characterise the fundamental organization of the system. Next, the development viewpoint supports the system's development process. Lastly, the deployment and operational viewpoints depict the system in its live environment.

The relative importance of each viewpoint depends on the type of information system. For example, for decision support systems, the context, information, and deployment viewpoints are the most important, whereas for a high-volume website the functional, development, and deployment views have the highest priority.

Figure 3: *Viewpoint groupings [9].*



3.1.2 Architectural perspectives

Although the combination of views can form a representation of the whole architecture, they are considered to be largely independent of one another. Quality attributes of a system, however, affect multiple or all of the aforementioned views, and can therefore not be considered as a separate view. Quality attributes are defined in definition 4, and need to be integrated into the existing views of a system to address the relevant concerns.

Definition 4 (Quality attribute)**.** *Quality attributes are measurable or testable properties of a system that are used to indicate how well the system satisfies the needs of its stakeholders [20].*

Rozanski and Woods (2011) [9] call a specific set of quality attributes an architectural perspective as described in definition 5. Examples of architectural perspectives are: availability, evolution, internationalisation, performance and scalability, security, and usability.

Definition 5 (Perspective)**.** *An architectural perspective is a collection of architectural activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties, that require consideration across a number of the system's architectural views [9].*

3.2 Software architecture reconstruction

Since a system's software architecture captures its most significant properties and design constraints, changes and additions made to a system that violate its architectural principles can degrade its performance and shorten its useful lifetime [10]. However, the rapidly changing stakeholder requirements and business conditions of a system, and with that the potential frequency and scale of software adaptations, make it difficult to control such architecture erosion. In other words, the realised architecture (see definition 6) drifts away from the intended architecture (definition 7). As this happens, architectural documentation often does not stay up to date (if present in the first place) and the people working with the system change over time [11]. At the same time, the architecture of a system cannot be re-established using solely the source code of a system, since it does not explicitly specify it [12].

Definition 6 (Realised architecture)**.** *The realised architecture refers to the architecture that is derived from source code.*

Definition 7 (Intended architecture)**.** *The intended architecture refers to the architecture that exists in human minds or in the software documentation.*

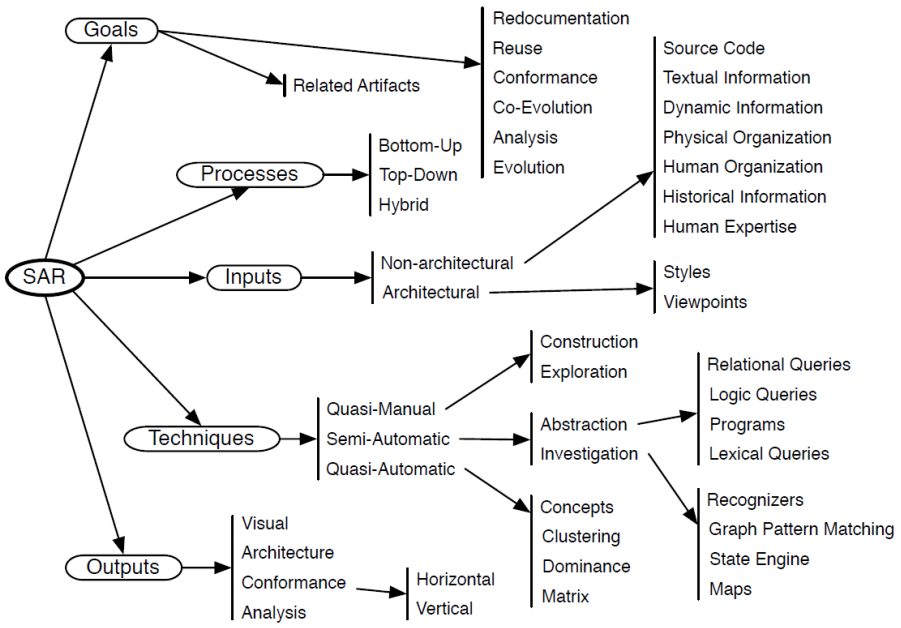
The aim of the field of Software Architecture Reconstruction (SAR) is to re-establish software abstractions. It can therefore be defined as:

Definition 8 (Software architecture reconstruction)**.** *Software architecture reconstruction (SAR) is a reverse engineering approach that aims at reconstructing viable architectural views of a software application [12].*

3.2.1 SAR state-of-the-art

Ducasse and Pollet (2009) [12] present the state-of-the-art in SAR approaches. An overview of their taxonomy is shown in Figure 4. They divide the field along five axes: goals, processes, inputs, techniques, and outputs. Since this chapter aims at answering the question "How can architectural information be extracted from a running software system?", the input axis is explained next.

Figure 4: *A process-oriented taxonomy for SAR [12].*



3.2.2 SAR inputs

Inputs of SAR can be both architectural and non-architectural. Architectural inputs can be viewpoints which are explained in section 3.1, and styles, which represent recurrent architectural situations such as data flow:

Definition 9 (Architectural style)**.** *An architectural style is a vocabulary of components and connector types, and a set of constraints on how they can be combined [21].*

However, to conduct meaningful SAR, non-architectural inputs are of vital importance. Reconstructing an architecture from only architectural inputs would mean to merely rewrite it. Non-architectural inputs for SAR as shown in Figure 4 are:

- **Source code:** Can be analysed as text, but mostly the meta-data of the code is used.
- **Textual information:** An example is comments in the source code, but these are mostly of poor quality [11].
- **Dynamic information:** Static information is often insufficient as it only provides limited insight into the run-time behaviour of a system. Dynamic information is extracted from a system execution and obtaining it is technically challenging.

- **Physical organization:** This refers to the source code's storage structure, which is almost always done in a tree structure. This structure can be used to identify connected code.
- **Human organization:** The structure of a system often reflects the communication structure of the organization [22].
- **Historical information:** This can be the version history of a project or other documents indicating how the system evolved over time. It is rarely used as an input for SAR, and only in (co-)evolution approaches.
- **Human expertise:** Although not as trustworthy as other inputs, it is essential for SAR. Examples are knowledge of business goals and design constraints.

3.3 Architecture mining framework

Van der Werf and Brinkkemper (2017) [1] propose the notion of Architecture Mining to close the gap between software architecture and the realised software system. They argue that techniques that aim at reconstructing the design, or even the architecture of a system such as SAR, mainly focus on functional aspects of a system and tend to ignore quality attributes. Approaches that do take quality attributes into account emphasise the reconstruction of architectural artefacts, rather than compliance checking of quality attributes during the operational phase of the software [2].

Many quality attributes are better assessed at run-time when the software system is in operation. By doing so, the architect gains insight into how the software is actually being used, which can serve to shape the next releases of the software product. For example, frequently used features can be grouped differently to improve overall performance.

This is why Van der Werf and Brinkkemper (2017) [1] advocate the use of run-time SED to assess the quality of software architecture. In this way, the realised software system can be monitored and analysed using data analytics techniques, which aids in making a step forward towards continuous architecting. Architecture mining can therefore be defined as:

Definition 10 (Architecture mining)**.** *Architecture mining is the collection, analysis, and interpretation of software execution data to foster architecture evaluation and evolution [1].*

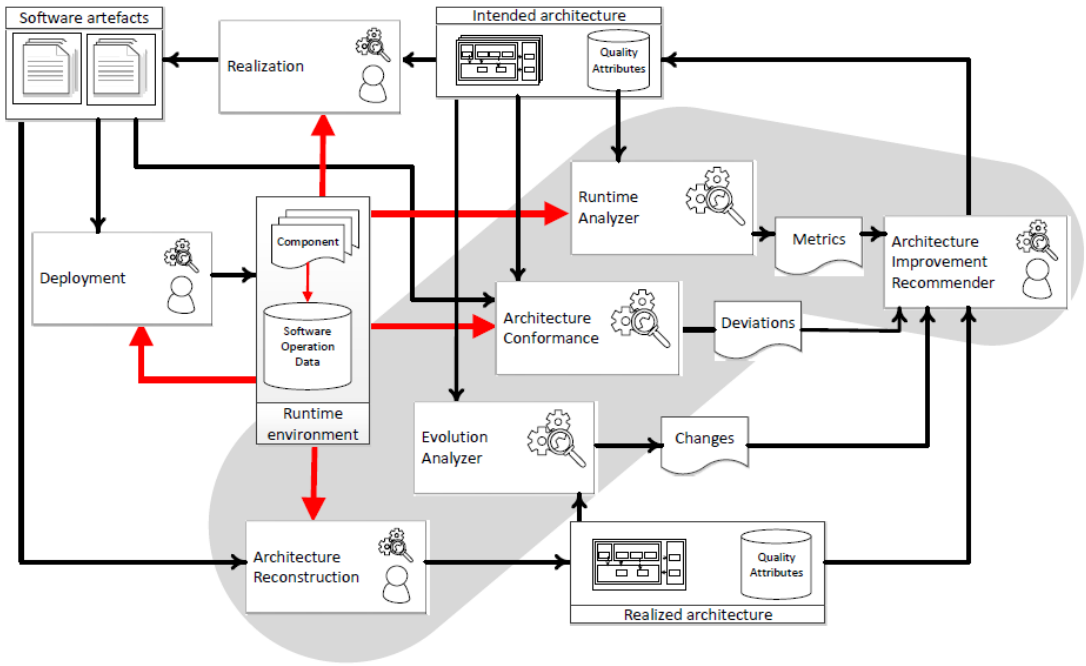
3.3.1 Conceptual overview

The conceptual overview of Architecture Mining is shown in Figure 5. It is a combination of the Software Operation Knowledge Framework, which is a method to drive the evolution of software products based on operation knowledge [23], with insights from process mining (see definition 11). It encompasses five main activities to support the architecture process which are listed below and will be explained next.

1. Software Architecture Reconstruction;
2. Architecture Conformance Checking;
3. Runtime Analyzer;
4. Evolution Analyzer;
5. Architecture Improvement Recommender.

Definition 11 (Process mining)**.** *Process mining is a technique that enables extracting knowledge by analysing event logs of information systems [24].*

Figure 5: *Architecture Mining Framework.*



In the framework shown in Figure 5, the architecture is shown as a set of structures which are represented by architectural views, and quality attributes that define quality constraints on the set of structures [20]. The black arrows indicate dependencies between the different elements, whereas the red arrows indicate which steps can benefit from SED. It should be noted that this does not imply a (fixed) order between the phases. The *intended* and *realised* architecture (see definitions 7 and 6) are separated to represent the notion of architectural erosion which is explained in section 3.2. System *realisation* encompasses many different *artefacts*, of which most are discussed in section 3.2.2, such as source code and documentation.

The intended architecture is used to derive the software artefacts from. These software artefacts are then used as an input for *Software Architecture Reconstruction* (activity 1, see definition 8) and *Architecture Conformance Checking* (ACC, activity 2, see definition 12). SAR results in a realised architecture, which together with the intended architecture and ACC results in a set of *deviations* from the intended architecture. The *Evolution Analyzer* (4) analyzes how the realised architecture drifted apart from the intended architecture, which results in a set of *changes* with respect to the intended architecture. Once the software artefacts are *deployed* and the system is operational in some *environment*, it starts collecting SED. Based on this data, the *Runtime Analyzer* (3) analyzes to which degree quality attributes specified in the intended architecture are satisfied. The output of the Runtime Analyzer is a set of *metrics*.

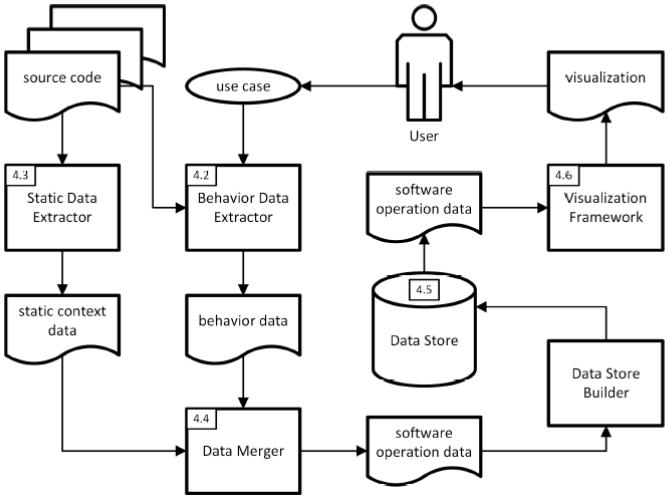
The derived realised architecture, metrics, deviations and changes are input for the *Architecture Improvement Recommender* (5), which ultimately results in an improved intended architecture.

Definition 12 (Architecture conformance checking)**.** *Architecture conformance checking (ACC) is a measure for testing to which degree the realised architecture conforms to the intended software architecture [25].*

3.4 Collecting software execution data

SED are logs that describe the operation of the software system [2]. Ipskamp (2018) [26] defines an information flow model to show how SED can be extracted from running systems, and converted into a suitable format in which it can be stored. This model is shown in Figure 6. The extraction of static and behaviour data as shown in this figure are explained below.

Figure 6: *Information flow model of SED [26].*



3.4.1 Extracting static data

To extract the required static data a *Static Data Extractor* can be used to obtain context data about the source code. Most available Static Data Extractors can be applied to multiple technologies. Examples of Static Data Extractors are Doxygen [27] and Sonargraph [28, 29]. Doxygen is a tool for generating documentation from source code which includes code structure and supports the programming languages: C++, C, Objective-C, C#, PHP, Java, and Python. Sonargraph is a static code analyzer that allows for monitoring software systems for technical quality, and enforcing rules regarding software architecture, metrics, and other aspects. Sonargraph supports Java, C#, Python 3, and C/C++.

3.4.2 Extracting behaviour data

Most software systems log events raised by the system [30]. These events can be at system or application level. Examples of system level events are warning, error or informational events. Examples of application level events are the completion and start of a functional element in the system. However, not all systems produce sufficient logging data, and therefore need to be enhanced to support the level of logging that is required [24]. When doing so, the architect should already consider during system design how quality attributes can be measured and which data needs to be collected [1]. Besides this, the use of Behaviour Data Extractor tools also depends highly on the system under analysis and its underlying technologies, version, and setup. As definition 13 shows, the behaviour of the system is the result of how the user interacts with it.

Definition 13 (Software behaviour)**.** *Software behaviour is the order of actions performed given a certain scenario i.e. a use case with all its context [26].*

The most common way of extracting software behaviour is to record a set of method or function calls by the user (the words method and function are interchangeable and depend on the programming language being used). This involves defining an execution scenario so that only the parts of interest of the software system are captured [31]. Some approaches for recording software behaviour record only the original call or method entry, which is the moment a method is called, while other approaches record both method entry and exit. Method exit is recorded either when the method returns or when it stops without returning anything. As described by Cornelissen (2009) [31], the benefits of dynamic analysis are precision and the use of a goal-oriented strategy. However, limitations are the inherent incompleteness (which is also true for software testing), difficulty of determining scenarios, scalability due to large amounts of data, and the observer effect i.e. the phenomenon in which software acts differently when under observation.

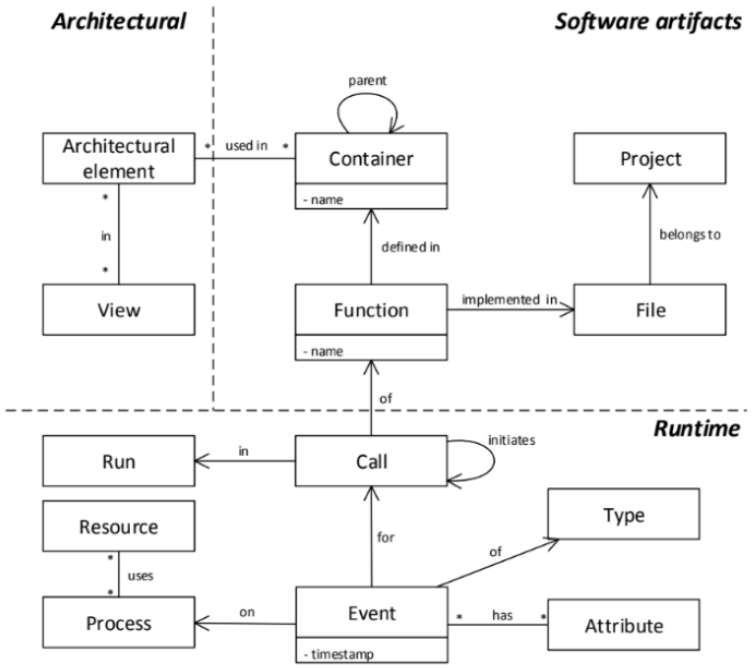
From a historical perspective, dynamic analysis was used for debugging, testing, and profiling [31]. The purpose of testing is the verification of correctness and profiling is used to measure (and optimise) performance. Debugging is not only used for locating faults, but also to understand the program under analysis. The use of dynamic analysis for program comprehension, as software became increasingly large and complex, thus originates from the discipline of debugging.

Many log analysis tools exist which can process terabytes of log data per day. Examples of this are Logentries [32], Loggly [33] and LogicMonitor [34]. However, these tools mostly log performance statistics such as server load and error rates. These measurements, though usable in conformance checking, offer little architectural insight.

3.4.3 Logging architectural information

The conceptual model of SED shown in Figure 7 can be used to validate whether the structural components of software can be derived from the gathered logging data. The model is divided into three parts: architectural, software artefacts and run-time. The first two represent structural elements and the latter represents the run-time aspects of software.

Figure 7: *Conceptual model of SED for Architecture Mining [1].*



Ritmeester (2018) [35] studies existing logging methods and accompanying tools to test whether they capture enough relevant information to be used for SAR (3.2). The tested tools are: Java Flight Recorder (JFR) for the collection of data in combination with Java Mission Control for its analysis, InTrace, and Java Platform Debugger Architecture (JPDA). The quality of the logs is determined based on five quality levels as proposed by Van der Aalst [24]. To reach the highest quality level, an event log has to be trustworthy, complete, and include well-defined events. Also, the events should be recorded in an automatic, systematic, reliable, and safe manner. Lastly, the events should have clear semantics and consider privacy and security adequately. In his study, Ritmeester (2018) [35] classifies JFR as level 2/3 (where level 1 is the highest quality level), and InTrace and JPDA as level 2. He notes that JPDA contains information on threads and method entry and exit location, which makes it the most useful tool for SAR out of the three.

An example of a log created by Ritmeester (2018) [35] with JPDA is shown in Figure 8. The log captures a run of the program under study, and in this run several methods/functions are called and executed such as “Hello.count” and “Hello.getname”. These method calls can occur through different events with different types (e.g. entry and exit), which in this case are method entry or exit. A method call is performed on a process which has one or more threads (see definition 14), in this case: “Thread-0” and “Thread-1”.

Definition 14 (Thread)**.** *A thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system [36].*

Figure 8: *Log created with Java Platform Debugger [35].*

```
-- VM Started --
1288003680765801 -- /main/start --
1288003692628961 -- /main/entry Hello.main(java.lang.String[])@Hello:20 -- {args:java.lang.String[]=[]}
1288003699369318 -- /main/| entry Hello.<init>(java.lang.String)@Hello:6 -- {naam:java.lang.String=H1}
1288003701016431 -- /main/| exit  Hello.<init>(java.lang.String)@Hello:8 -- {naam:java.lang.String=H1}
1288003702044441 -- /main/| entry Hello.<init>(java.lang.String)@Hello:6 -- {naam:java.lang.String=H2}
1288003702714042 -- /main/| exit  Hello.<init>(java.lang.String)@Hello:8 -- {naam:java.lang.String=H2}
1288003703359011 -- /Thread-0/start --
1288003703470680 -- /Thread-0/entry Hello.run()@Hello:27 -- {}
1288003704140692 -- /Thread-0/| entry Hello.count(int)@Hello:16 -- {i:int=0}
1288003705577606 -- /main/exit  Hello.main(java.lang.String[])@Hello:22 -- {args:java.lang.String[]=[]}
1288003706355592 -- /Thread-1/start --
1288003706461924 -- /Thread-1/entry Hello.run()@Hello:27 -- {}
1288003706959917 -- /Thread-1/| entry Hello.hoera(int)@Hello:16 -- {i:int=0}
1288003707637319 -- /main/end --
1288003710000427 -- /Thread-1/| | entry Hello.getName()@Hello:11 -- {}
1288003710705335 -- /Thread-1/| | exit  Hello.getName()@Hello:11  -- {}
1288003711185265 -- /Thread-0/| | entry Hello.getName()@Hello:11 -- {}
1288003711649593 -- /Thread-0/| | exit  Hello.getName()@Hello:11  -- {}
1288003713873935 -- /Thread-1/| exit  Hello.count(int)@Hello:17 -- {i:int=0}
1288003714456091 -- /Thread-1/| entry Hello.count(int)@Hello:16 -- {i:int=1}
1288003715048099 -- /Thread-1/| | entry Hello.getName()@Hello:11 -- {}
1288003715514480 -- /Thread-1/| | exit  Hello.getName()@Hello:11  -- {}
1288003716891044 -- /Thread-0/| exit  Hello.count(int)@Hello:17 -- {i:int=0}
1288003717525749 -- /Thread-0/| entry Hello.count(int)@Hello:16 -- {i:int=1}
```

As can be seen in Figure 7, the connection between the run-time elements and software artefacts is the call of a method. In this case, the line on which the method is positioned within the source code is also shown: “@Hello: 20”. The file in which the source code is stored belongs to a project, of which neither are shown in this log. The organization of methods/functions depends on the programming language used, which is why a generic container data type is shown in the model. A container can for example be a class, package, or namespace in which the method is defined. In the log this information is captured when the method is called: “Hello” in “Hello.count” is the container. The information that is missing in this log are the callers of the methods, i.e. the log shows which method is called, but not what called the method. Method and class/object hierarchy of a system can therefore not be determined based on this log.

De Jong (2019) [8] fills this gap by creating a non-intrusive tool that outputs logs consisting of method calls with their callers and callees registered. The tool is called AjpoLog, which stands for AspectJ Partially Ordered Logging, as it uses AspectJ to instrument systems [37]. As an example, a piece of a log created by AjpoLog is shown in Figure 9. The first two columns show the start and end time of the method being called, from which the duration of the method execution can be derived. In the third column the thread on which the method is executed is shown. The fourth and fifth column show the identifiers of the calling object and the object that contains the method being called. The last column shows the fully qualified name of the method being called. To give an example, on line 1 of the log shown in Figure 9, the class "Band" (caller) calls the method "getName()" (message), which is part of the class "Song" (callee).

Figure 9: *Piece of a log created with AjpoLog (after conversion to csv format) [8].*

Start Time	End Time	Thread	Caller	Callee	Message
2019-03-26T16:20:02,333	2019-03-26T16:20:02,333	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	org.architecturemining.program.example.band.Song.CallerPsuedoId: 1205555397	public java.lang.String org.architecturemining.program.example.band.Song.getName()
2019-03-26T16:20:02,333	2019-03-26T16:20:02,334	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	java.lang.StringBuilder.CallerPsuedoId: 1543974463	public java.lang.StringBuilder java.lang.StringBuilder.append(java.lang.String)
2019-03-26T16:20:02,334	2019-03-26T16:20:02,334	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	java.lang.StringBuilder.CallerPsuedoId: 1543974463	public java.lang.String java.lang.StringBuilder.toString()
2019-03-26T16:20:02,334	2019-03-26T16:20:02,335	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	java.io.PrintStream.CallerPsuedoId: 1293618474	public void java.io.PrintStream.println(java.lang.String)
2019-03-26T16:20:02,335	2019-03-26T16:20:02,335	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	java.util.HashSet.CallerPsuedoId: 897074030	public abstract java.util.Iterator java.util.Set.iterator()
2019-03-26T16:20:02,336	2019-03-26T16:20:02,337	[main]	org.architecturemining.program.example.band.Band.CallerPsuedoId: 278934944	java.util.HashMap.KeyIterator.CallerPsuedoId: 156545103	public abstract boolean java.util.Iterator.hasNext()

3.4.4 Merging static and behaviour data

When both static context data and behaviour data are extracted from a system, these need to be merged by a *Data Merger* to allow for efficient storage and interfacing. To do this, both datasets need to have one or more attributes through which they can be linked. In object-oriented programming, the fully qualified name of a function is often a good property to link on. If this name is not unique, it can be combined with the file path of the source file to create a unique identifier for the function.

3.5 Conclusions

Software architecture plays an important role in various aspects of software development [19]. Rozanski and Woods (2011) [9] argue that to portray the elements of the architecture that are relevant, architectural views can be used. These are representations of a set of system elements and relations associated with them. Within these views, architectural perspectives can be integrated to ensure that a system exhibits a particular set of related quality properties.

The field of SAR aims at reconstructing architectural views of software when the intended architecture (see definition 7) has drifted from the realised architecture (definition 6) due to architectural erosion. Ducasse and Pollet (2009) [12] present the state-of-the-art in SAR approaches. In their framework, both architectural inputs such as viewpoints, and non-architectural inputs such as source code are important.

Van der Werf and Brinkkemper (2017) [1] argue that techniques that aim at reconstructing the design or architecture of a system such as SAR, mainly focus on functional aspects of a system and tend to ignore quality attributes. Approaches that do take quality attributes into account emphasise the reconstruction of architectural artefacts, rather than compliance checking of quality attributes during the operational phase of the software [2]. This is why run-time SED are most suitable to evaluate and evolve software architecture.

SED are a combination of static context data and behaviour data, and can be collected by applying static and behaviour data extractors on a running system. A variety of static data extractor tools can be used to obtain the needed context data. However, of the studied dynamic data extractor tools, only one is suitable for architecture mining: AjpoLog. This is because this is the only tool studied so far that captures not only the method being called (callee), but also the calling object (caller). To merge static and behaviour data into SED, in object-oriented programming, the fully qualified name of functions can be used. These conclusions answer the sub-research question: **SQ 1. How can architectural information be extracted from a running software system?**

4 Visualisation Techniques and Tools

This chapter aims at answering the sub-research question:

SQ 3. How are visualisation techniques and supporting tools currently used to represent software architecture and its evolution over time?

First the four types of SAV techniques as identified by Shahin, Liang, and Babar (2014) [5] are explained. Second, characteristics of techniques and tools that should be considered when designing them are set forth. Next, the techniques and tools that different types of stakeholders of the system require are described. Last, best practices found in the field of information visualisation are summarised.

4.1 Types of visualisation techniques

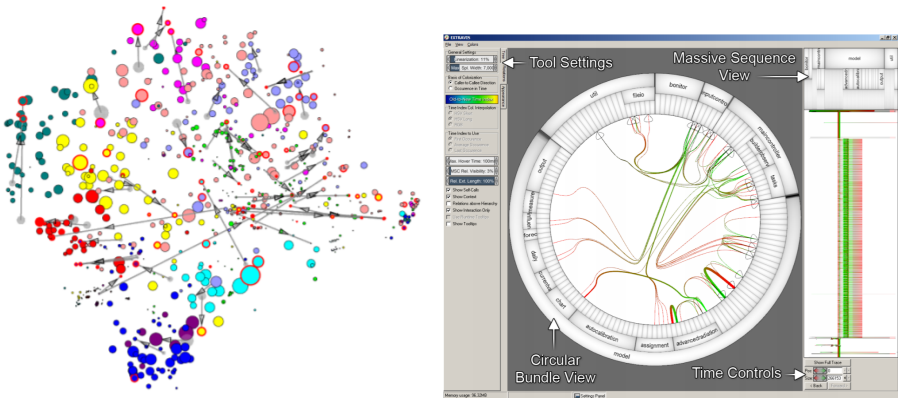
Graduleva and Adibi Dahaj (2017) [38] interviewed project managers, architects, and developers to identify their current usage of SAV techniques and corresponding tools. They found out that most interviewees used UML diagrams to support their work. For architects this were mainly component, state machine and sequence diagrams, and for developers class diagrams. Additionally, architects used a few graph-based diagrams when UML failed to visualise a high level of architecture.

Shahin, Liang, and Babar (2014) [5] systematically review and classify the visualisation techniques and associated tools reported for software architecture, and how they have been assessed and applied. They identify four types of visualisation techniques used in the architecting process: graph-based, notation-based, matrix-based, and metaphor-based visualisation. These four types of techniques are now described using several examples.

4.1.1 Graph-based techniques

Of the four techniques, graph-based visualisation is most popular in industry [5]. Graph-based visualisations use nodes and links to represent the structural relationships between architecture elements. This puts more emphasis on the overall properties of a structure than on the types of nodes. Two examples of this are shown in Figure 10. In geometry, nodes and links are called vertices and edges. Throughout this report vertices and edges will be referred to as nodes and links, because this is the jargon used in the literature about visualisation techniques that this report refers to. For clarity, all four definitions from the Oxford dictionary are provided below.

Figure 10: *Examples of graph-based techniques [39, 40].*



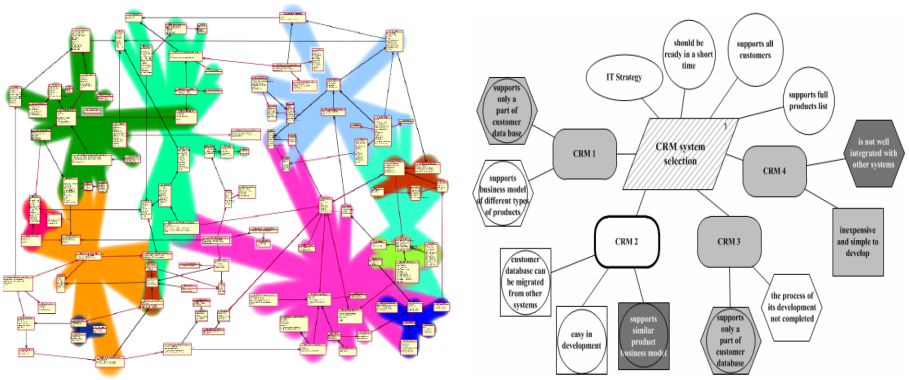
Definition 15 (Vertex/node)**.** *In geometry, a vertex is a meeting point of two lines that form an angle [41]. A node is a point in a network or diagram at which lines or pathways intersect or branch [42].*

Definition 16 (Edge/link)**.** *The outside limit of an object, area, or surface [43]. In computing, a link is a code or instruction which connects one part of a program or an element in a list to another. [44].*

4.1.2 Notation-based techniques

This category of visualisations is the second-most popular [5]. It is a combination of three modeling techniques: unified modeling language (UML), systems modeling language (SysML), and specific notation-based visualisation. UML is an industry standard and general-purpose visual modeling language to specify, design, construct, and document software-intensive systems [45]. SysML is built on UML and reuses and extends it for system engineering applications, including software, hardware, and information systems [46]. Notation-based visualisations represent the relationship between and role of elements in a structure using various notations. The name and type/label of the nodes and links are therefore of major importance in contrast to graph-based notations in which the overall properties of a structure are more important.

Figure 11: *Examples of notation-based techniques [47, 48].*



4.1.3 Matrix-based techniques

Matrix visualisation is a graphical technique that can simultaneously explore the associations of up to thousands of subjects, variables, and their interactions, without first reducing dimension [49]. Matrix-based visualisation can support a graph-based visualisation by giving complementary information when the graph is large or dense [5]. Only 8 % of the studies reviewed by Shahin et. al used this technique. Two examples of matrix-based techniques are shown in Figure 12.

4.1.4 Metaphor-based techniques

This category uses familiar physical world contexts (such as cities) to visualise SA entities and their relationships [52]. There is an increasing tendency to utilize real metaphors as opposed to abstract ones as a means to amplify cognition. However, opinions are divided when it comes to the effectiveness of these visualisations. According to Balzer et. al (2004) [53], the use of metaphors makes the visualisation process particularly intuitive and effective. On the other hand, Carpendale et. al (2008) [4] state that there is no empirical evidence of the added benefits of these real metaphors, and thus it is yet to be scientifically justified. Examples of this technique are shown in Figure 13.

Figure 12: *Examples of matrix-based techniques [50, 51].*

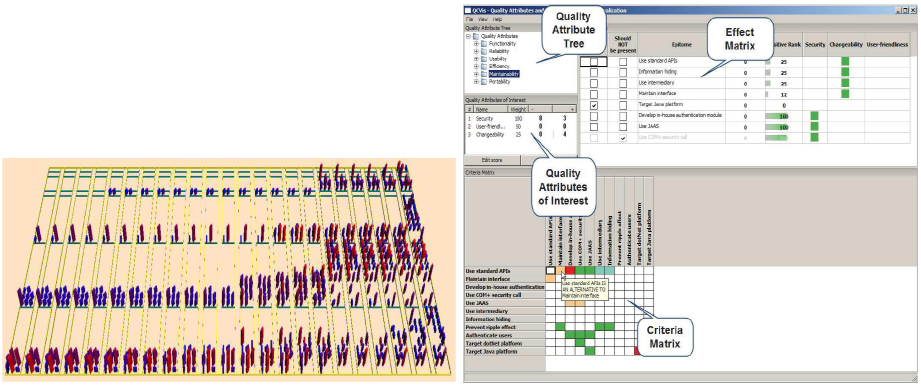
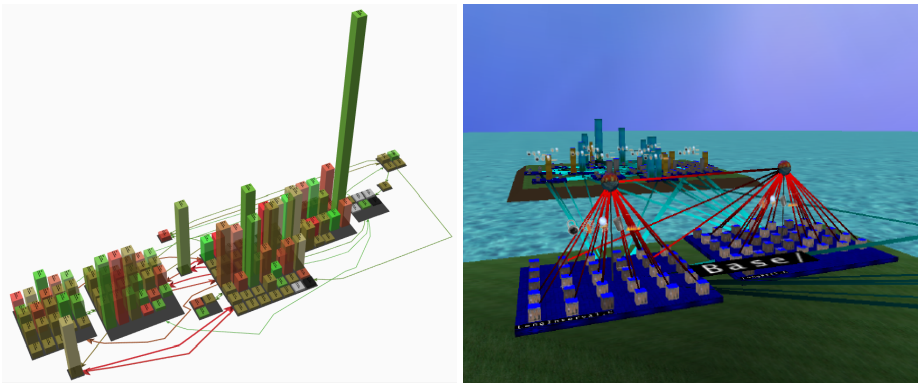


Figure 13: *Examples of metaphor-based techniques [54, 55].*



4.2 Characteristics of techniques and tools to consider

4.2.1 Multiplicity of view

Carpendale et. al (2008) [4] identify two schools of thought regarding multiplicity of view. The first school asserts that any visualisation should support multiple views of the architecture, at different levels of detail, to satisfy the audience's different interests. The other school of thought claims that a carefully designed single view of the visualisation might be more effective and meaningful in conveying multiple aspects of the architecture than a multiple view approach. They think a single view can provide different levels of detail, in the sense that the user can draw their own mental maps based upon it.

As mentioned before, Gallagher et. al (2008) [56] derive and construct a qualitative framework, with seven key areas and 31 features, for the assessment of SAV tools. The authors claim multiple views of software architecture are a requirement for a good visualisation tool. Additionally, a representation of the viewpoint definition should be displayed.

4.2.2 Dimensionality

Munzner (2014) [18] states that 3D visualisation should not be used without the right justification. When the user's task involves understanding the shape of inherently three-dimensional structures, the benefits of 3D certainly outweigh the costs. In that case, of which an example is studying human anatomy, designers can use the many existing interaction idioms to mitigate the costs of using 3D. In all other contexts, a careful justification is needed.

Munzner also notes that most tasks involving abstract data do not benefit from 3D. For example, a study tested subjects' ability to quickly locate web page images, and showed that

this ability deteriorated as their freedom to use the third dimension increased. This was true for both real-world physical models as well as an equivalent computer-based virtual system. Their subjective responses also indicated that they found the 3D interfaces more cluttered and less efficient [57].

Mostly, rather than choosing a visual encoding using three dimensions of spatial position, a better answer is to visually encode using only two dimensions of spatial position [18]. An appropriate 2D encoding usually follows from a different choice of data abstraction, where the original dataset is transformed by computing derived data. Cues that convey depth information to our visual system include: occlusion, perspective distortion, shadows and lighting, familiar size, and stereoscopic disparity. These should be analysed before making the decision to use a 3D encoding of data.

According to Carpendale et. al (2008) [4], the main reason for using 3D visualisation in the field of software architecture is that the representation of highly dimensional data can cause occlusion using 2D or even 1D visualisations. However, the distinction between 2D and 3D representations refers to the dimensionality of the graphical visuals, not the dimensionality of the data itself. Therefore, a carefully designed 2D representation of an architecture should be capable of representing more than two dimensions in the dataset. This conforms to the arguments by Munzner.

4.2.3 Medium

Merino et. al (2017) [58] study the impact of the medium in the effectiveness of 3D software visualisations. They deploy 3D city visualisations across a standard computer screen, an immersive 3D environment, and a physical 3D printed model. They show that even though developers using a 3D printed model required the least time to identify outliers, they perceived less difficulty when using visualising systems based on a standard computer screen. Notably, developers using an immersive 3D environment obtained the highest recollection. This emphasises the need of the explicit inclusion for the medium and technique as properties for benchmarks that evaluate software visualisations.

4.2.4 Interactivity

In addition to their other recommendations, Gallagher et. al (2008) [56] also prescribe the types of interactivity a visualisation should be able to support. First, the user should be able to browse the visualisation by following concepts within the visualisation. Second, the user should be able to search the visualisation for arbitrary architectural information. Next, query drilling architectural information should be possible. This means the user should be able to search the data space and then recursively search within the resulting dataset. Finally, inter- and within- view navigation should be possible.

Munzner (2014) [18] argues that there is always a trade-off between finding automatable aspects of a visualisation, and relying on the human using the visualisation to detect patterns. The benefit of interaction is that people can explore a larger information space compared to a single static image. Despite this, interaction requires the time and attention of the human using the visualisation. A best practice in this regard is to automatically detect features of interest to explicitly bring to the user's attention via the visual encoding.

4.2.5 Implementation

In the study of Graduleva et. al (2017) [38], both project managers and architects used hand-drawn or manually controlled tools more often than automatic tools. Developers used automatic tools only slightly more often than hand-drawn tools. The main reason for not using automatic tools was a lack of IDE-integrated tools which are able to generate readable diagrams from a large number of entities quickly. An ideal automatic tool would substitute a collection of tools, and generate scalable diagrams that visualise the composition and relations between entities at different levels of abstraction. To compress and improve readability, "details-on-demand" by for instance filtering and searching would be useful. Based on this, the authors argue that graph-based hierarchical edge bundles, semantic dependency matrices, clustered graph layout, and/or edge evolution filmstrip techniques can be used.

Of the analysed studies by Shahin et. al (2014) [5], 42 percent provide automatic tool support for the used visualisation techniques, 47 percent semi-automatic tool support, and 11 percent manual tool support. They note that (semi-)automatic tool support improves the practical applicability of visualisations by practitioners.

For the implementation of SAV tools, Gallagher et. al (2008) [56] prescribe that automatic generation of the visualisations should be possible. Besides that, it is important to consider platform dependence. If platform choice prohibits remote capture of system data, the visualisation should be able to execute on the same platform as the software it is intended to visualise. Remote capture may be preferred for its potential in reducing unwanted interaction with the software. As there are many stakeholders of a software system, there may also be a one-to-one mapping of role to physical user. The visualisation should therefore also be able to support multiple users concurrently and asynchronously.

4.2.6 Data representation

Gallagher et. al (2008) [56] prescribe which features an SAV tool should have to appropriately represent static and dynamic data. To begin with, it should support an appropriate set of static and dynamic data sources, large volumes of data, and a multitude of software architectures. Next, recovery of architectural information from sources that are not directly architectural is preferable. Also, dynamic events should be associated with elements of software architecture, and dynamic data should be collected in a non invasive way. Finally, live collection of data should be possible as well as recording dynamic data for subsequent replay.

4.3 Designing and evaluating information visualisations

The field of visual analytics builds better and more effective ways to understand and analyse large datasets [3]. The main reason to use visualisation of data is that the human brain processes images 60,000 times faster than text, and 90 percent of information transmitted to the brain is visual [13]. Card, Mackinlay, and Shneiderman (2009) [59] define information visualisation as:

Definition 17 (Information visualisation)**.** *Information visualisation (InfoVis) is the use of computer-supported, interactive visual representations of data to amplify cognition.*

4.3.1 Designing information visualisations

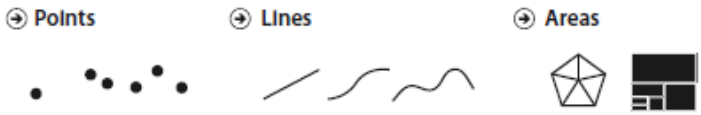
Munzner (2014) [18] describes how to effectively design the marks and channels that compose a visualisation. According to her, the core of the design space of visual encodings can be described as an orthogonal combination of two aspects: graphical elements called marks (see definition 18), and visual channels to control their appearance (see definition 19).

Definition 18 (Mark)**.** *A mark is a basic graphical element in an image i.e. geometric elements that depict items or links [18].*

Definition 19 (Visual channel)**.** *A visual channel is a way to control the appearance of marks, independent of the dimensionality of the geometric primitive [18].*

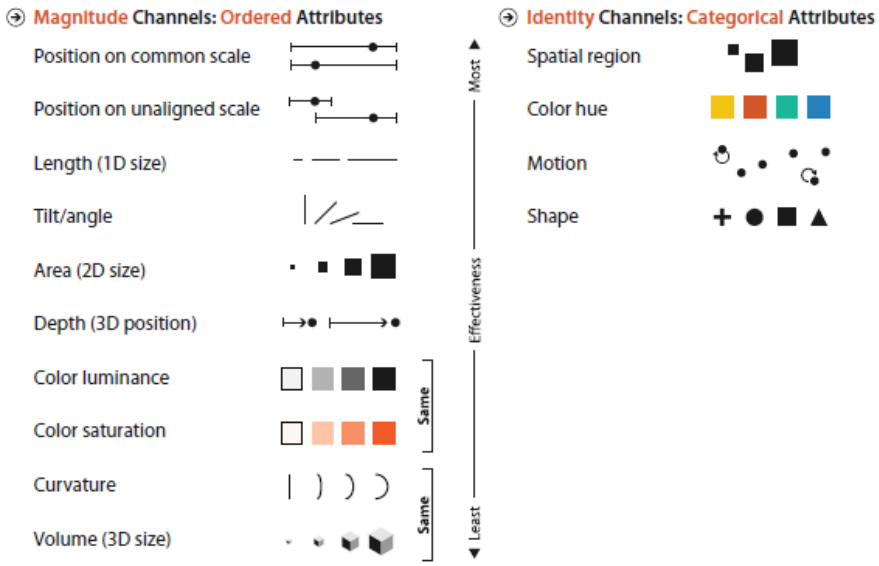
Marks are classified according to the number of spatial dimensions they require as shown using examples in Figure 14. A zerodimensional (0D) mark is a point, a one-dimensional (1D) mark is a line, and a two-dimensional (2D) mark is an area. A threedimensional (3D) volume mark is possible, but they are not frequently used. There are two link mark types: (1) connection, which shows a pairwise relationship between two items, using a line, and (2) containment, which shows hierarchical relationships using areas by nesting connection marks within each other at multiple levels (Munzner does not mention overlap of 2D areas as a link mark type).

Figure 14: *Marks as geometric primitives [18].*



Examples of the many visual channels that can encode information are: colour, shape, position, tilt, size, and volume. The human perceptual system has two fundamentally different kinds of sensory modalities. Identity channels tell us information about what something is or where it is, and magnitude channels tell us how much of something there is. Not all channels are equal: the same data attribute encoded with two different visual channels will result in different information content. Therefore, the use of marks and channels is guided by two principles: expressiveness and effectiveness.

Figure 15: *Channels ranked by effectiveness according to data and channel type [18].*



The expressiveness principle states that the visual encoding of a visualisation should express all of, and only, the information in the attributes of the dataset. This means that if one of the visual encodings does not have an attribute mapped to it, it violates the expressiveness principle. The most important expression of this principle is that ordered data should be shown in a way that our perceptual system intrinsically understands as ordered. At the same time, unordered data should not be shown in a way that perceptually implies an ordering that does not exist. The identity channels are the correct match for the categorical attributes that have no intrinsic order. The magnitude channels are the correct match for the ordered attributes, both ordinal and quantitative.

The effectiveness principle prescribes that the importance of an attribute should match the salience of the channel, that is, its noticeability. In other words, the most important attributes should be encoded with the most effective channels in order to be most noticeable, and then decreasingly important attributes can be matched with less effective channels. Figure 5.6 shows the effectiveness rankings for each visual channel. The effectiveness of channels is defined according to accuracy, discriminability, separability, the ability to provide visual popout, and the ability to provide perceptual groupings.

4.3.2 Evaluating information visualisations

Evaluation is a key research challenge within the information visualisation community. Forsell and Johansson (2010) [60] present a set of general heuristics to evaluate common and important usability problems in information visualisation techniques. They evaluated the 63 heuristics of 6 earlier published heuristic sets, by letting subjects use them to explain a collection of 74 usability problems derived from earlier InfoVis evaluations. The resulting set provided the highest explanatory coverage, and contains the following ten heuristics:

- **Information coding:** Perception of information is directly dependent on the mapping of data elements to visual objects. This should be enhanced by using realistic characteristics/techniques or the use of additional symbols.
- **Minimal actions:** Concerns workload with respect to the number of actions necessary to accomplish a goal or a task.
- **Flexibility:** Flexibility is reflected in the number of possible ways of achieving a given goal. It refers to the means available to customization in order to take into account working strategies, habits and task requirements.
- **Orientation and help:** Functions like support to control levels of details, redo/undo of actions and representing additional information.
- **Spatial organization:** Concerns users' orientation in the information space, the distribution of elements in the layout, precision and legibility, efficiency in space usage and distortion of visual elements.
- **Consistency:** Refers to the way design choices are maintained in similar contexts, and are different when applied to different contexts.
- **Recognition rather than recall:** The user should not have to memorize a lot of information to carry out tasks.
- **Prompting:** Refers to all means that help to know all alternatives when several actions are possible depending on the context.
- **Remove the extraneous:** Concerns whether any extra information can be a distraction and take the eye away from seeing the data or making comparisons.
- **Data set reduction:** Concerns provided features for reducing a data set, their efficiency and ease of use.

4.4 Conclusions

To conclude this chapter, a concrete answer to sub-research question 3 is formulated:

SQ 3. How are visualisation techniques and supporting tools currently used to represent software architecture?

Shahin, Liang, and Babar (2014) [5] systematically review and classify the visualisation techniques and associated tools reported for software architecture, and how they have been assessed and applied. They identify four types of visualisation techniques used in the architecting process; graph-based, notation-based, matrix-based, and metaphor-based visualisation. 42 percents of the analysed studies by Shahin et. al (2014) [5] provide automatic tool support for the used visualisation techniques, 47 percent semi-automatic tool support, and 11 percent manual tool support.

In the studied articles, several characteristics of techniques and tools to consider when designing SAVs are identified: multiplicity of view [4, 56], dimensionality [4], medium [58], interactivity [56], implementation [5, 56], and data representation [56].

Since the studied articles are written by researchers in the field of software visualisation, additional research in the field of visualisation analysis and design is consulted. Section 4.3 provides an overview of best practices in the field of visualisation design, which can be used as a reference during the treatment design phase. In this analysis, the work of Munzner et al. [18] is emphasised due to the high citation count of the work and perceived usefulness of the research.

5 Stakeholder Requirements

Since the requirements of SAV depend highly on the needs of the stakeholder who is using the visualisation (see defintion 20), this section describes the possible stakeholders of SAV and their needs. The chapter starts by describing the various purposes SAV can have, after which the possible tasks SAV can support are set forth per type of stakeholder. In this way, this chapter aims to answer the question:

SQ 2.Which questions should visualisations of software dynamics be able to answer in order to be valuable to system stakeholders?

Since (project) managers [61, 55, 62, 63], architects [61, 55, 56, 4], and developers [61, 55, 63, 4] are mentioned most often in the reviewed articles, these stakeholder groups are selected to discuss in more detail in sections 5.2, 5.3, and 5.4. However, it should be noted that Gallagher et. al (2008) [56] also identify testers, operators, designers, development managers, sales, field support, and system administrators as stakeholders. Carpendale et. al (2008) [4] add customers to this list, and Panas et. al (2007) [55] view maintainers as a group separate from developers. Finally, technical users and consultants are mentioned as stakeholders by Telea et. al (2010) [62].

Definition 20 (Stakeholder)**.** *A stakeholder in the architecture of a system is an individual, team, organization, or classes thereof, having an interest in the realisation of the system [9].*

5.1 Purposes of Software Architecture visualisation

According to Diehl (2007) [64], the main motivation for using software visualisation in general is to help stakeholders understand different aspects of software systems during the software development process, and to reduce the cost of software evolution. To create an SAV, entities in a software system domain are mapped to their graphical representation to aid comprehension and development [65]. Software architecture visualisation (SAV) is defined as:

Definition 21 (Software architecture visualisation)**.** *Software architecture visualisation is a visual representation of architectural models, and some or all of the architectural design decisions about the models [66].*

Shahin, Liang and Babar (2014) [5] identify ten categories of purposes for using visualisation in the field of software architecture based on a systematic literature review of papers published between 1 February 1999 and 1 July 2011. The most frequently reported purposes identified by them are:

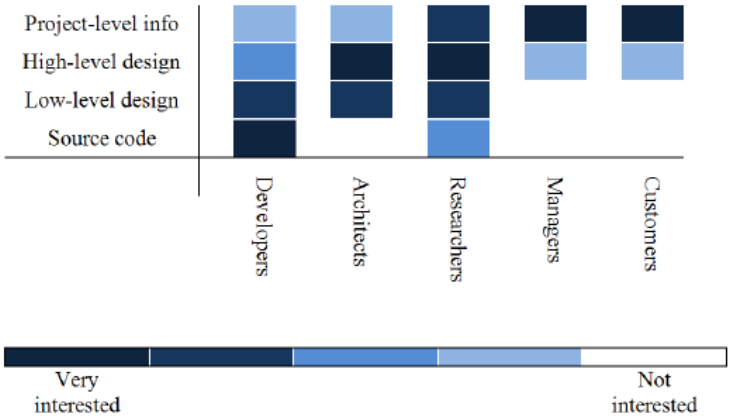
1. Improving the understanding of architecture evolution (26% of reviewed papers) .
2. Improving the understanding of static characteristics of architecture (24%).
3. Improving search, navigation, and exploration of architecture design (24%).
4. Improving the understanding of architecture design through design decision visualisation (21%).
5. Supporting architecture re-engineering and reverse engineering (13%).
6. Detecting violations, flaws, and faults in architecture design (11%).
7. Providing traceability between architecture entities and software artefacts (11%).
8. Improving the understanding of behavioural characteristics of architecture (6%).
9. Checking compatibility and synchronization between architecture design and implementation (6%).

10. Supporting for model-driven development using architecture design (2%).

A systematic literature review of articles on software visualisation from 2010 to 2015 describes several purposes software visualisations can have in general [61]. Most of the reviewed articles are related to software structure, behaviour, and evolution. The authors argue that software visualisation researchers should consider if there is synergy between the goals and visual means and tools that could be applied to understand software, software process and communication. The review also notes that collaboration and engagement are mentioned often as goals of visualisations. For SAV in particular this is supported by Gallagher et. al [56] (2008), who claim that a good SAV should aid communication of the architecture to intended stakeholders.

Carpendale and Ghanam (2008) [4] show the groups of stakeholders along with the level of detail in software architecture they are interested in (Figure 16).

Figure 16: *Various levels of interest of the software visualisation audience [4].*



5.2 Project Manager Requirements

As can be seen in Figure 16, Carpendale and Ghanam (2008) [4] claim that managers are mostly interested in the project-level information of a software system. They want to monitor the progress of the project and determine the completion of the envisioned development goals. Visualisations can be useful in determining whether the next deadline will be met, and how to allocate team members to each part of the system [55]. This is in line with the definition of a project manager:

Definition 22 (Project manager)**.** *A project manager is the person in overall charge of the planning and execution of a particular project [67].*

Telea et. al (2010) [62] also claim that it could be useful to managers if an SAV supports the monitoring of the evolution i.e. progress of a system over time. In this way, trends such as architectural erosion, rule violation, and quality decay can be identified. Managers are not interested in individual lines of code or call relations, but rather in coarser detail levels such as file, class, and user (author). Therefore, they need visualisation techniques with a high level of abstraction that can simultaneously display numerous attributes or metrics. Examples of this are parallel coordinates, dense pixel charts, tree maps, and timelines. Another study suggested the use of clustered graph layout for this purpose [38].

Besides this, Panas et. al (2007) [55] pose that managers could use visualisations to identify components of a system that need improvement, such as components with high maintenance costs. This is confirmed by a case study conducted by Graduleva and Adibi Dahaj (2017) [38], who interviewed managers of software system projects. They found that manager's requirements towards SAV are mostly the high-level representation of the composition of systems, and relationships between systems and subsystems. Moreover, they found visualisation useful when communicating, making decisions, and understanding architecture. Especially visualisation of system implementation in relation to requirements would be useful to them. This is in line with the research by Cleland (2013) [68], who argues that visualisations can aid the understanding, and then the communication of system architecture to a variety of project stakeholders.

5.3 Software Architect Requirements

As shown in Figure 16, architects are most interested in the high-level design of a system, followed by a significant interest in the low-level design and a slight interest in the project-level information. Architects find it important to realise the different characteristics of the architecture they design, such as complexity, coupling, cohesion, and other attributes [4]. They are not interested in the source code on its own. Rozanski and Woods (2011) define the role of a software architect as follows:

Definition 23 (Software Architect)**.** *The architect is responsible for designing, documenting, and leading the construction of a system that meets the needs of all its stakeholders [9].*

Shahin, Liang and Babar (2014) [5] identify the architecting activities that are supported by the SAV techniques as found in the reviewed studies. They based this classification on the proposed architecting activities by Li et al. (2013) [69], which compose the entire architecture life-cycle and are supported by the general architecting activities. The activities "architecture understanding" and "architecture description" as proposed by Li et al. are excluded, since visualisations are by nature expected to support these.

As can be seen in Figure 17, the most common architecting activities supported by visualisation techniques are architecture recovery, evolution, and evaluation. The other identified architecting activities are: change impact analysis, architectural analysis, architectural synthesis, architectural implementation, and architecture reuse.

Figure 17: *Distribution of architecting activities supported by visualisation techniques [5].*

Architecting activity	Number of studies
Architecture recovery (AR)	25
Architectural evolution (AEV)	16
Architectural evaluation (AE)	11
Change impact analysis (CIA)	10
Architectural analysis (AA)	9
Architectural synthesis (AS)	7
Architectural implementation (AI)	5
Architecture reuse (ARU)	4

Interviews of Graduleva and Adibi Dahaj (2017) [38] with both system and design architects show that architects require information about (1) composition of and relations between (clusters of) systems, subsystems, classes, and components, (2) most used or "problematic" components, (3) relationships between systems, classes and components, and (4) implications of new flows to old flows. In this study, they were the stakeholders with the highest demand for additional information and metrics, such as types of signals, implications of a new flow to the old ones, implementation in relation to requirements, and test coverage.

5.4 Developer Requirements

As shown in Figure 16, developers are interested in all levels of abstraction, but mostly in the lowest ones. They focus on understanding the system [4, 5, 6], while monitoring code changes and their impact [4]. Rozanski and Woods (2011) define the role of a developer as:

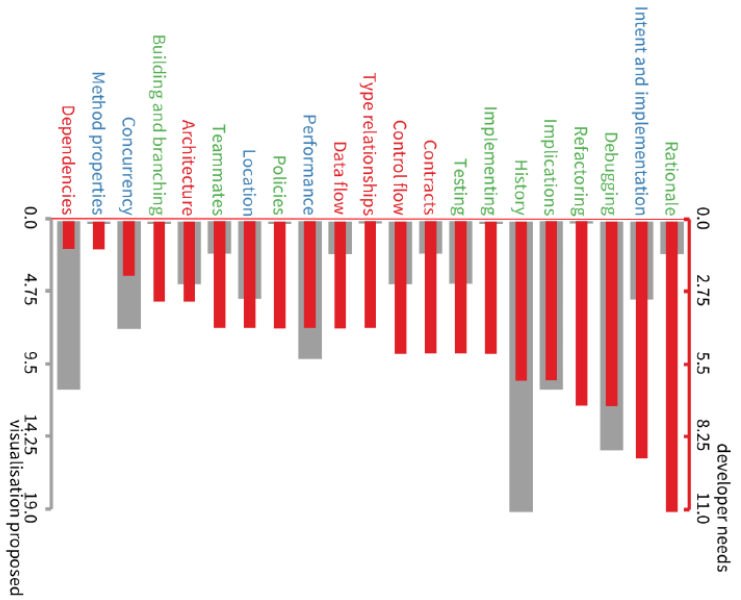
Definition 24 (Software Developer)**.** *A developer constructs and deploys a system from specifications (or leads the teams that do this) [9].*

This is confirmed by Telea et al. (2010) [62], who state that developers are concerned with a low level of abstraction just like testers and maintainers. They studied 23 developers using four different visualisation tools, and all of them found correlated views of code, metrics, structure, and dependencies indispensable. Techniques they are interested in are therefore similar to those in which architects are interested. Examples of this are tree-maps and hierarchically bundled edges. These would be able to represent clutter-free layouts of thousands of entities and relationships with zero user intervention.

When studying the requirements of developers, Graduleva et. al (2017) [38] observe that information about the composition and relationships between classes and packages are useful to them. Other information needed, but currently lacking or not supported by the techniques they used, included types of signals, CPU heavy parts of the code, and revision history.

LaToza et al. (2010) [70] categorised the problem domains that developers deal with as follows: (1) changes, (2) elements, and (3) element relationships. Merino, Ghafari, and Nierstrasz (2016) [6] estimate the importance of these problem domains for practitioners. They claim that the more types of questions a problem domain contains, the more important that domain is for developers. Figure 18 shows the results of their study. In this figure, the problem domains are shown on the y-axis and encoded according to the category they belong to: "changes" are coloured green, "elements" are coloured blue, and "element relationships" are coloured red. In the figure, the importance of developer needs (red bars) is compared to the number of visualisation techniques that address these needs (grey bars). From this analysis, the authors learned that practitioners are mostly concerned about changes, while existing visualisation distribute their attention among all three categories. Because of that, problem domains that are very important to developers have little visualisation support, and less important domains receive a good degree of attention.

Figure 18: *Comparing the degree of importance of developer needs vs. their visualisation support by problem domain. [6].*



5.5 Conclusions

In this chapter, sub-research question 2 is answered by analysing related work:

SQ 2. Which questions should visualisations of software dynamics be able to answer in order to be valuable to system stakeholders?

According to Diehl (2007) [64], the main motivation for using software visualisation in general is to help stakeholders understand and comprehend different aspects of software systems during the software development process, and to reduce the cost of software evolution. The stakeholders that are identified most often in the reviewed articles are project managers [61, 55, 62, 63], architects [61, 55, 56, 4], and developers [61, 55, 63, 4].

Managers are interested in project-level information [4]. The information that is useful to them is about quality related trends, system evolution, system implementation in relation to requirements, composition of and relations between (sub-)systems, and most used or "problematic" components [38]. Examples of tasks that this information can support are: allocating team members and costs [55], monitoring progress [62, 4], determining deadlines [4], communicating [68], decision-making [38], and understanding the system's architecture [38, 68].

Software architects are most interested in the high- and low-level design information of a system [4]. This is mainly information about system implementation in relation to requirements, composition of and relations between (sub-)systems, composition of clusters of classes, most used or "problematic" components, relations between classes and components, and implications of old flows to new flows [38]. This information can support them in the tasks identified in Figure 17.

Developers are interested in all levels of abstraction, but mostly in the lowest ones [4]. Developers find correlated views of code, metrics, structure, and dependencies indispensable [62]. The problem domains they encounter can be put into three categories: changes, elements, and element relationships [70]. When comparing the importance of developer needs in these problem domains to the number of visualisation techniques that address those needs, discrepancies can be found as shown in Figure 18. Graduleva et. al (2017) [38] observe that their information need in the relationships and composition categories concerns classes and packages. Other information needed includes: types of signals, CPU heavy parts of the code, and revision history.

6 Case Study Interview Results

This section summarises the most important and relevant results of the interviews, which were conducted with the DevOps team of the system to which the visualisations will be applied to. As described in chapter 2, though this data has poor external validity [15], it will give some insight into how SAV is currently used at the company, how the envisioned visualisations can be valuable to the system's stakeholders, and a means to validate the dashboard design. By doing so, it aims at answering sub-research question 2 for the specific context in which the envisioned visualisations will be applied:

SQ 2. Which questions should visualisations of software execution data be able to answer in order to be valuable to system stakeholders?

Three interviews were held, one with the DevOps Engineer in the team with the most experience with the system [A.1.2 Quote 1], one with the Software Architect of the system [A.2.2 Quote 4], and one with the Product Owner of the system. These interviewees are from here on referred to with their position title as specified by themselves in the interviews. The interview protocol is provided in Appendix A. An overview of the quotes from the interviews this section is based on can be found in Appendices A.1, A.2, and A.3.

The correspondence of answers given by interviewees was checked against reality by studying the team during their daily work. One of the researchers worked in the same room as the team for six months and was therefore able to observe the team's work process.

6.1 System Under Study

The system under study is part of an application landscape that is used in the Netherlands as a central information system for 7,300 users. It has 1.7 millions registrations, and every month 170,000 manual and 1.2 million automatic checks are done in the system. It is an application landscape that consists of three applications which are all written in the JAVA programming language [A.2.1 Quote 1], and make use of cloud computing services [A.2.1 Quote 4]. From now on these applications will be referred to as application 1, 2, and 3. The three applications are actually composed of several sub-applications, for example application 1 contains six sub-applications [A.2.1 Quote 1 and 2].

The applications are connected to several supporting applications such as a mail server [A.2.1 Quote 3] and a support portal in which tickets are registered [A.2.1 Quote 5]. For development specific purposes there is also an issue management system called JIRA. In this system progress can be tracked such as which functionalities have to be build, and which bugs need to be solved [A.2.1 Quote 6].

So far the development of the system has been about transitioning it to the current supplier and resolving the issues that this caused. Application 1 is the oldest application in the landscape originating from 2006, and consists of around 225,000 lines of code. The second oldest application is application 2 [A.2.1 Quote 7]. Application 3 was added last and is used by only around 40-50 people. This makes it a lot less traffic intensive than the other two applications [A.1.1 Quote 1]. The first time the system was run in production by the DevOps team was in December 2018 [A.1.1 Quote 8]. Despite the trouble of transitioning it from the previous supplier, the moment that new functionality can be developed instead of just resolving issues is coming close according to one of the interviewees [A.1.1 Quote 2].

6.2 Stakeholder Roles and Tasks

The system's team consists of six members whose roles and experience are specified in Table 1. The team members all work on all three applications comprising the application landscape, which is described in the previous section (6.1). At the moment of the interviews, there was a database administrator working on the system one day a month, and another DevOps Engineer who worked on the system one day a week. However, these team members have left as of May 2019 [A.3.1 Quote 13].

Figure 19: *Simplified representation of the work process of the DevOps team.*

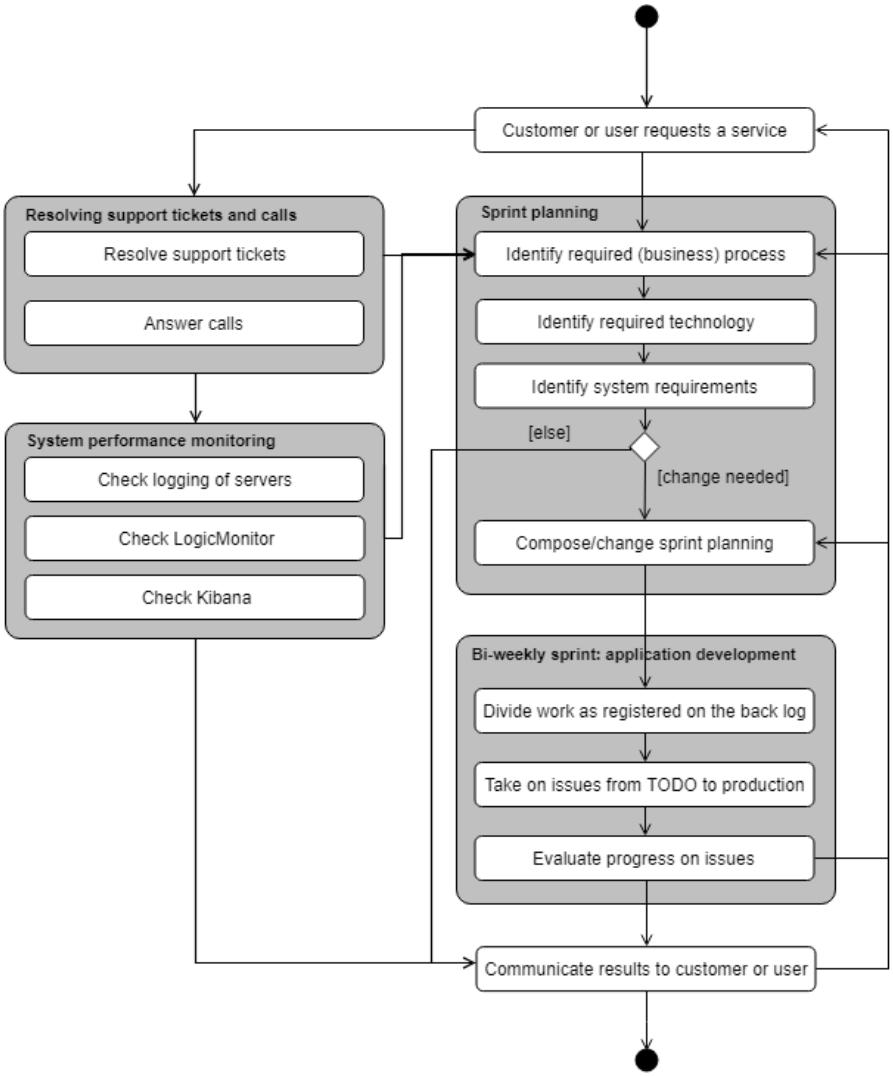


Table 1: Members of the system's DevOps team and their experience (on April 2019).

Role	**FTE**	**Experience with the system in months**	**Previous experience in a similar role in years**	**Experience with software design tools**
Product Owner	1	16	4	Yes
Software Architect	0.6	18	20	Yes
DevOps Engineer 1	1	12	2.5	No
DevOps Engineer 2	1	3	6	Yes
DevOps Engineer 3	1	9	0	No
DevOps Engineer 4/ Scrum Master	0.8	2	0	No

The software design method that is used to develop the system is called DevOps, which means that every person is expected to be able to do both development and operations tasks [A.1.2 Quote 2]. In Figure 19, a simplified version of the work process of the team is shown based on the UML Activity Diagram technique [45] and explained next.

Based on the interview results, four categories of activities are identified and shown in the figure in grey blocks: (1) resolving support tickets and calls, (2) system performance monitoring, (3) sprint planning, and (4) bi-weekly sprint: application development. The process starts when the customer or a user of the system has a request. This can be either a support ticket [A.1 Quote 15], a phone call [A.3.1 Quote 17], requests to one of the system servers such as the web proxy [A.1.5 Quote 9], or a new service the customer would like to provide to its end users [A.2.2 Quote 19].

6.2.1 Resolving support tickets and calls

If the request is a support ticket or a call, the team usually has to explain to a user how the system works, or resolve an issue that arises with the system [A.1.2 Quote 17]. This can in turn result in checking the logs of the servers for errors and warnings [A.1.2 Quote 13]. For example, the interviewed DevOps Engineer uses the e-mail address of the customer certificate and the time the problem occurred to trace in the logs where in the system it went wrong [A.1.4 Quote 1]. All members of the team take on these system support tasks one day a week [A.3.1 Quote 15]. The reason this decision was made is so that everyone keeps a good understanding of what is going on, what operations is struggling with, and which processes could be optimised because they cause recurring issues [A.3.1 Quote 12].

Table 2: *Tasks of the category "resolving support tickets and calls" per role within the DevOps team.*

Tasks / role	**Product Owner**	**Software Architect**	**DevOps Engineer**	**Scrum Master**
Resolve support tickets	x	x	x	x
Answer calls	x	x	x	x

6.2.2 System performance monitoring

User requests within the application can result in alarm bells going off in the office of the team [A.1.2 Quote 10]. In that case, the team checks the performance metrics shown by the LogicMonitor tool to identify what is going on [A.1.5 Quote 5]. LogicMonitor is explained in more detail in section 6.5. The team plans to also start using Kibana in the future [A.1.4 Quote 10, [71]]. They intend to use Kibana to follow what happens with a request from the moment it enters the system up until it is completed [A.1.4 Quote 9].

Table 3: *Tasks of the category "system performance monitoring" per role within the DevOps team.*

Tasks / role	**Product Owner**	**Software Architect**	**DevOps Engineer**	**Scrum Master**
Check logging of servers	x	x	x	x
Check LogicMonitor	x	x	x	x

6.2.3 Sprint planning

The Product Owner is the main person in the team who does the direct coordination with the customer [A.2.2 Quote 16]. If the customer wants to provide a new service to its participants, the Product Owner, Software Architect, and customer come together to identify the process that is needed to realise this service [A.2.2 Quote 16, 20 and 21]. Once that is clear, the Product Owner and Software Architect identify how the system can support that process, and what is needed to integrate the new parts of the system in the current system infrastructure [A.2.2 Quote 16, 20 and 22]. This results in a set of identified system requirements that they translate to a sprint planning [A.2.2 Quote 23]. One sprint is a set of tasks that can potentially be finished in two weeks [A.3.1 Quote 23]. The planning of the sprints happens in collaboration with the customer, to prioritise the issues according to her wishes [A.3.1 Quote 24].

Table 4: *Tasks of the category "sprint planning" per role within the DevOps team.*

Tasks / role	**Product Owner**	**Software Architect**	**DevOps Engineer**	**Scrum Master**
Identify customer requirements	x			
Identify required (business) process	x	x		
Identify required technology		x		
Identify system requirements		x		
Compose/change sprint planning	x	x		x

6.2.4 Bi-weekly sprints: application development

As mentioned before, the team uses bi-weekly sprints for which issues/tasks are registered on a so called backlog, which need to be completed at the end of the two weeks [A.1.2 Quote 3]. At the end of a sprint, the issues that are solved or not solved are evaluated to see how the work process can be improved [A.1.2 Quote 4]. The Scrum Master in the team takes the lead in this, and monitors the progress of the sprint, so that imperfections are solved and team members have the resources they need [A.3.1 Quote 20].

The DevOps Engineers in the team are the ones who take issues as registered on the sprint backlog [A.1.2 Quote 16] from TODO to production [A.2.2 Quote 18]. To manage the sprint planning a tool called JIRA is used [A.1.2 Quote 5]. This tool is also used to assign tasks [A.1.2 Quote 6]: in JIRA "you can drag an issue to active or in progress and then you can mark yourself as executor" [A.1.2 Quote 7]. Examples of tasks that the DevOps Engineers perform to solve issues are: writing and editing source code [A.1.2 Quote 12], application maintenance [A.1.2 Quote 9], server maintenance [A.1.2 Quote 8], and infrastructure deployment / networking [A.1.2 Quote 10 and A.1.2 Quote 11]. Solving issues can take between five minutes and two hours [A.1.2 Quote 14].

Table 5: *Tasks of the category "bi-weekly sprint" per role within the DevOps team.*

Tasks / role	**Product Owner**	**Software Architect**	**DevOps Engineer**	**Scrum Master**
Divide work as registered on the back log			x	
Take on issues from TODO to production			x	
Evaluate progress on issues			x	x

6.3 Sources of information about the system

Besides the manual of the system minimal documentation is available [A.1.3 Quote 5 and A.3.2 Quote 4]. There are also some comments in the code that describe what the code does. Despite this, the names of entities in the code are clear and therefore of good quality [A.1.3 Quote 6]. The table below shows which sources of information the interviewees mentioned using to gain understanding of the system. As can be seen, the source code of the system is leading together with the experience of the Software Architect (“colleague(s) with more experience with the system") [A.2.3 Quote 4 and A.3.2 Quote 5]. The Software Architect of the system is also the only one who mentioned using tools to gain a deeper understanding of the source code. An example of this is Sonarcube, which is a static analysis tool that is mainly used to detect bad practices [A.2.3 Quote 8]. The architect does note that it is not very useful for getting an overview of how the system is composed. For that he drew diagrams himself which will be shown and explained in section 6.5.

Table 6: *Sources of information used to gain understanding of the system [A.1.3, A.2.3, and A.3.2].*

Tasks / role	**Product Owner**	**Software Architect**	**DevOps Engineer**
Logs on server performance and requests	x	x	x
LogicMonitor	x	x	x
Manual of the system			x
Phone calls with the customer			x
Source code		x	x
Own memory			x
Colleague(s) with more experience with the system	x		x
Diagrams with system overview		x	
Sonarcube		x	
Oracle Cloud metrics: CPU, disk usage, networks etc.		x	

6.4 Information needs

The information needs as mentioned by the interviewees can be translated to questions they would like an answer to. As can be seen in Table 7, it would be useful to the team to have a better overview of relations between objects/methods in the source code, so they have a better way to communicate about the system and identify where pieces of code are located. This could additionally be of use when the team transfers the monolithic architecture the system has now to a more micro-service like architecture [A.2.4 Quote 9]. By for example identifying closely coupled components, they can depict which things they would like separate in the application.

Second, information about the process requests follow through the system could help identify bottlenecks, "dead" code, and the methods exceptions originate. If this information would be available, the team could focus more on development instead of operations in the future.

Table 7: *Information needs of the interviewed stakeholders [A.1.4, A.2.4, and A.3.3].*

No.	Information needs / role	Product Owner	Software Architect	DevOps Engineer
1	What are the relations between objects/methods in the system?	x	x	x
2	Which path does a request follow through the system? (what is the chain of objects and methods)	x	x	x
3	Which objects/methods are called most often?	x	x	
4	Which methods take a long time to execute? (bottlenecks)	x		x
5	Which objects are closely coupled together?	x	x	
6	In which methods do exceptions originate?	x		x
7	What are the dependencies of the system on third-party libraries?		x	
8	Where are design flaws and poor code quality?	x	x	
9	Which code seems to never be executed? ("dead code")	x	x	
10	How often are features used? ("dead features")		x	x

6.5 Current visualisation tools and techniques

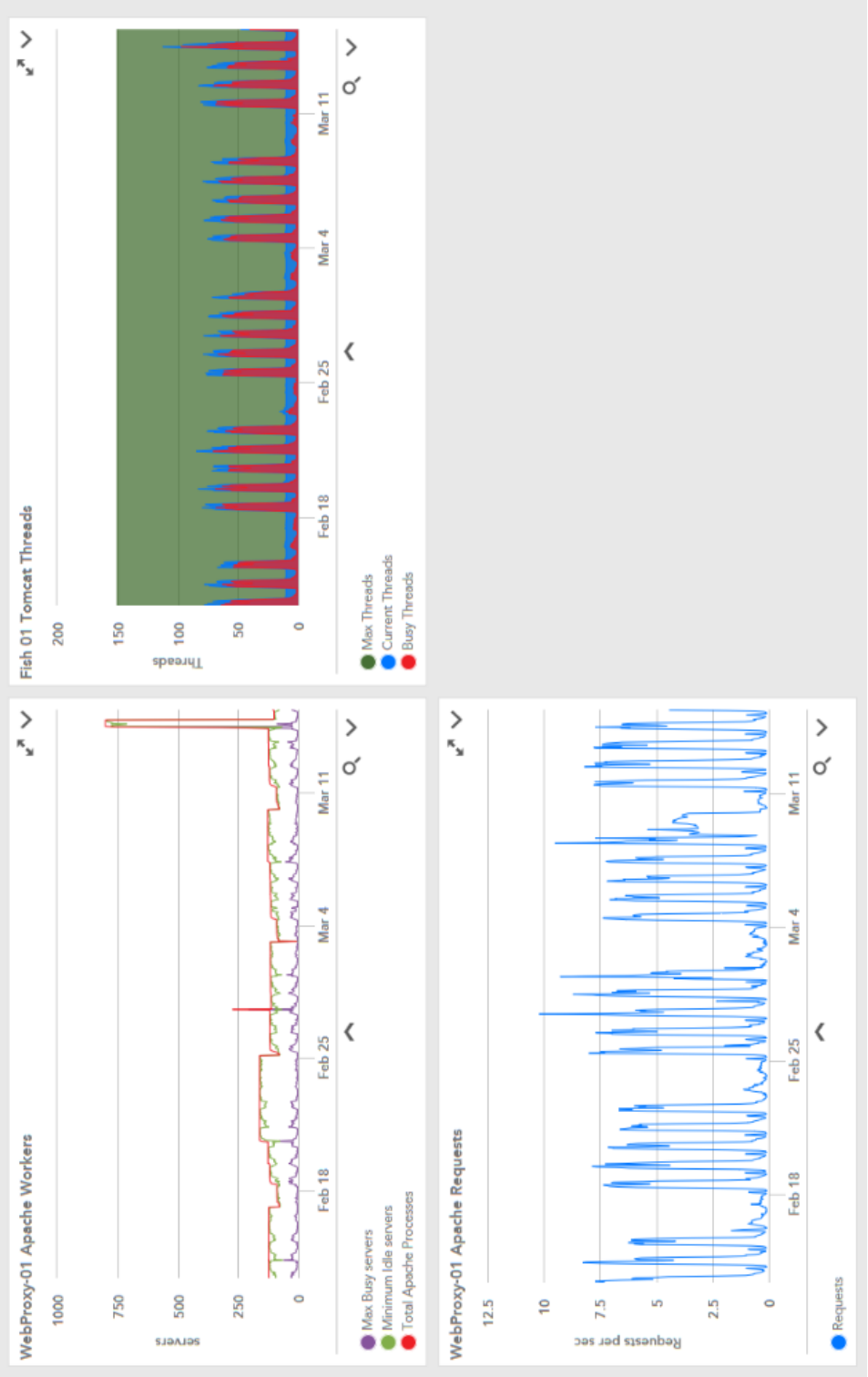
6.5.1 LogicMonitor

The only visualisation of the system that the Product Owner and DevOps Engineer use is shown in Figure 20. The three graphs in this figure are used to visualise how many requests enter the system [A.1.5 Quote 1], how much traffic there is [A.1.5 Quote 2], and how many workers are busy and available in the web proxy/server [A.1.5 Quote 3 and 4].

The visualisations are generated by a tool called LogicMonitor, which is the only monitoring system that the team uses at the moment [A.1.5 Quote 8]. It is coupled to the system under study and automatically retrieves the required information from it [A.1.5 Quote 9]. The DevOps Engineer finds LogicMonitor very useful to monitor the status of the servers. It however misses a way to monitor application specific things. If the Tomcat server reacts normally it for example does not notice when an application is not working [A.1.5 Quote 11].

In the graph at the top left of Figure 20, the green line represents the number of free workers and the purple line the number of busy workers. Therefore, the green line is usually above the purple one which means the workers can handle all the request that come into the system. If those lines turn around or cross something is going on [A.1.5 Quote 5]. When that happens, peeps and buzzers go off in the office so the team is alerted [A.1.5 Quote 10].

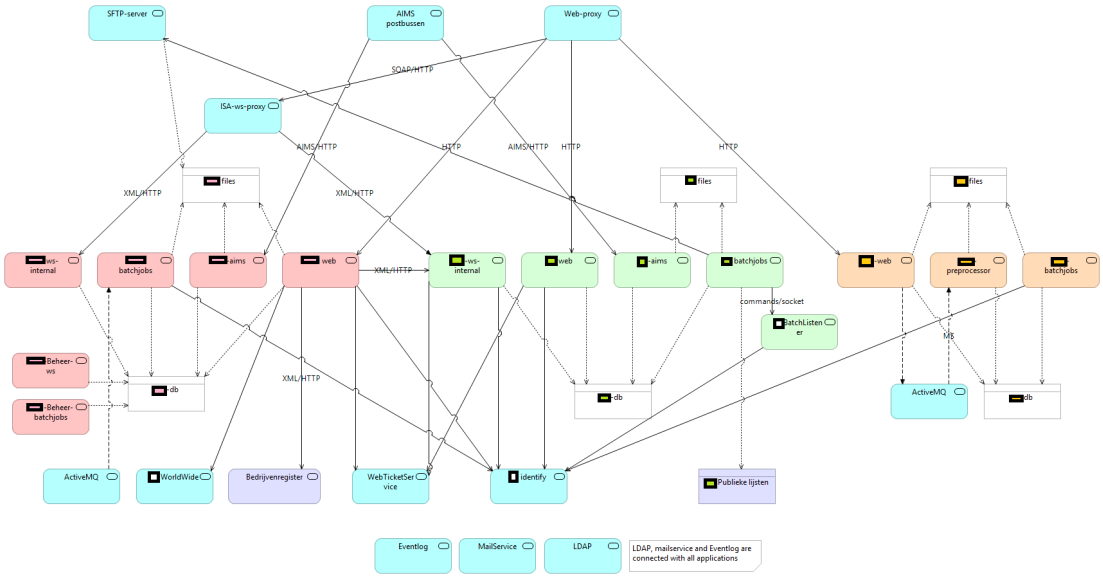
Figure 20: *LogicMonitor graphs.*



6.5.2 System Overview

Besides LogicMonitor, the Software Architect of the system also uses some diagrams that give an overview of the application landscape [A.2.5 Quote 1]. He created these using ArchiMate, which is an open-source tool in which the business, application, and infrastructure layers can be modelled [72]. To give an impression of what this looks like, the used diagram to provide a high-level overview is shown in Figure 21. The picture shows what the functional applications are and how they collaborate [A.2.5 Quote 2]. Besides this picture, there are some pictures that internally to the functional applications describe which libraries are used and how that is composed [A.2.5 Quote 3].

Figure 21: *Image called "Application Interfaces" as used by the DevOps team.*



6.6 Conclusions

This chapter answers sub-research question two for the specific system under study:

SQ 2. Which questions should visualisations of software dynamics be able to answer in order to be valuable to system stakeholders?

The questions identified by holding interviews with three different types of system stakeholders are shown in Table 7.

7 Case Study Survey Results

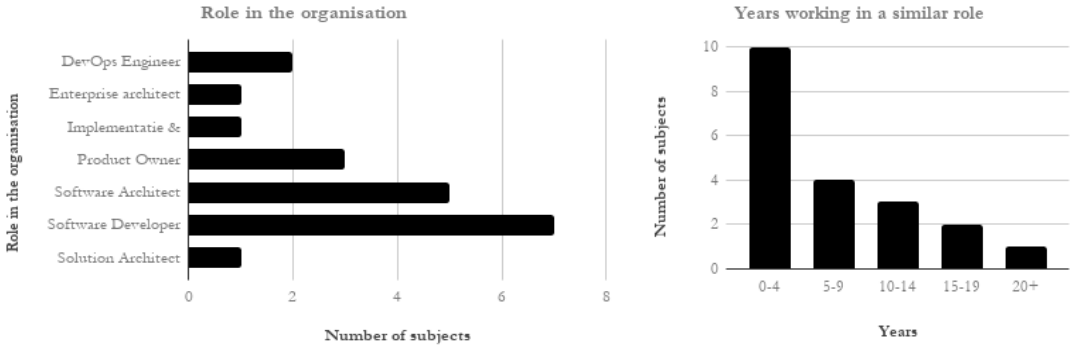
To prioritise the stakeholder requirements the envisioned dashboard of visualisations has to fulfil, a survey was carried out. The survey can be found in Appendix A. It was sent out to IT experts who work with Java systems, within the company at which the case study was executed. A total of 20 experts responded, including all six members of the DevOps team of the system the visualisations will be applied to.

7.1 Subjects

The survey was sent out through Google forms between May 10 and May 28 2019. Seven out of 20 respondents were Software Developers, five were Software Architects, and 3 were Product Owners. Additionally, one Solution Architect, one Enterprise Architect, one Implementation and Migration Consultant, and two DevOps Engineers responded. Three of the DevOps Engineers of the DevOps team in which the case study is executed called themselves Software Developers when asked in the survey. The team member who was identified as the architect of the system called himself a DevOps Engineer. This shows that there is an overlap in the definitions of a Software Developer and a DevOps Engineer, and a Software Architect and a DevOps Engineer. This should be taken into account when interpreting the results.

Half of the respondents had 0-4 years of working experience in a similar role when the survey was conducted. Four had 5-9 years of experience, three 10-14, two 15-19, and one 20+.

Figure 22: *Position and experience of survey participants.*



7.2 Ranking of information needs

The table below shows the average ranking that the respondents assigned to the different questions. It stands out that when Product Owners rank the questions, the question they find most important is: "Where are design flaws and poor code quality?". Software Architects rank the question: "What are the dependencies of the system on third-party libraries?" much higher than the other types of stakeholders (5.0 versus 6.0 and 7.4). This would be their fourth most important question. Software Developers rank the question: "In which methods do exceptions originate?" very highly (3.9 versus 5.7 and 7.6).

Table 8: *Information needs of the interviewed stakeholders [A.1.4, A.2.4, and A.3.3].*

No.	Information needs / role	All (20)	Product Owners (3)	Software Architects (5)	Software Developers (7)
1	Which path does a user request follow through the system? (what is the chain of classes and methods)	2.9	4	2.2	2.0
2	Which methods take a long time to execute? (bottlenecks)	3.9	4.3	2.4	4.6
3	Which classes/methods are called most often?	5.3	5.3	4.2	6.0
4	Where are design flaws and poor code quality?	5.5	**3.0**	6.0	6.3
5	What are the relations between classes/methods in the system?	5.9	5.7	5.4	6.0
6	What are the dependencies of the system on third-party libraries?	6.0	6.0	**5.0**	7.4
7	Which code seems to never be executed? ("dead code")	6.1	6.7	6.8	5.6
8	In which methods do exceptions originate?	6.2	5.7	7.6	**3.9**
9	Which classes are closely coupled together?	6.5	7.7	7.0	7.1
10	How often are features used? ("dead features")	6.9	6.7	8.4	6.1

7.3 Reasons for Ranking

To gain more insight into the reasons why the respondents ranked the ten proposed questions as shown in Table 8, four additional questions were asked of which the results will now be discussed.

7.3.1 Are there questions in the list above that you can already quickly answer for the system(s) that you work with today? How? Which sources of information do you use to answer these questions?

As an answer to this question, the respondents named a number of tools and techniques that they use to answer the questions in Table 8.

One of the DevOps Engineer notes that in his team they are implementing the so called ELK Stack (Elasticsearch, Logstash, and Kibana) to extract information from their logs such as: user access, errors logs, and stack traces. One Product Owner mentions that they only know the time of the total user request through monitoring from log files, not the path it follows through the system. Another Product Owner confirms this by noting that they only know the relations between total software components or network/cloud components, for example how many times the system calls the database or multiple databases.

One Software Developer notes that dead code is already highlighted by the IDE (Integrated Development Environment) that he uses. Another Software Developer notes that he uses the tool called Jacoco to identify dead code (if unit tests are properly build). He also notes that Java stack-traces already provide a good way to show the origin of exceptions.

One Software Architect mentions that he generates UML class diagrams of the systems he works with. One Software Developer also mentions that class diagrams are a means to display relations between classes, but notes that automatically generated class diagrams are very difficult to interpret for complex programs.

The static source code analysers mentioned by the respondents are: PMD, FindBugs, and Sonarqube. The respondents use these technologies to discover static dependencies of the system and analyse design (flaws) and code quality. Other techniques used are the Maven dependency trees and POM files, that show the dependencies of an application in a Maven based application. One of the Software Architects also notes that the Oracle technology they use includes a dependency view for databases.

One Product Owner mentions that he primarily relies on the team he works with for information about software quality. The Enterprise Architect that responded notes that the most important source of information to him are the stakeholders. In this case this would be the administrators, testers, and end users. He uses this to compose a process and an information model if required and to describe the dependencies of the system.

7.3.2 Consider the question in the list above that you rated as most needed (1)? How would you make use of the answer to this question? What value does it have to you?

Six participants rated question number 1 as most important. The first reason that is given for this is that it can help understand how functionality can be changed and how monitoring can be set up for the highest value chains. Second, it could provide knowledge of the system while tracing and debugging problems. Third, it could explain how data flows through the system, though it should be noted that this participant viewed user requests as messages. Fourth, it could answer multiple questions at once, and knowing the path within the code would also give insight in what parts are used often or not. The last reason given is that it would make it easier to understand programs and isolate problems.

Out of the 20 subjects, four rated question 2 as being most important. The reasons provided are that it would be easier to identify where the "quick wins" are. Additionally, it would be useful to identify where performance efforts should be focused, and since better performance would mean better user experience the customers would be more satisfied. Moreover, it would be useful to optimise code and analyse whether the architecture should be adapted. Finally, it would pinpoint which parts of the code slow down user requests, which can support investigating why it is slow and making changes to it, or tuning the settings to gain performance. There might be badly implemented code or wrong architecture.

None of the participants ranked questions 3 and 5 as most needed. One Product Owner rated question 4 as most needed, because according to him it predicts future costs and failure the most. The Solution Architect did so too, because he thinks this information would help define the quality of custom made software systems, how they are structured, and whether modules/components are decoupled.

Two participants ranked question 6 as most needed. The reasons for this were that according to them, the dependencies of a system are a major factor in pinpointing how complex an application is in a maintenance environment. Only by fully understanding these dependencies can the application be maintained. It would also be required to solve performance issues.

Both DevOps Engineers rated question 7 as most needed. One of them rated the question as such because it could help remove waste from the system and make the context of the system smaller and easier to understand. The second engineer simply states "we probably have a lot of dead code".

One Software Developer ranked question 8 as number one, to know which errors are most common and improve the code that causes the exceptions. Another developer ranked question 9 as such, because it would help limiting the number of bottlenecks or even make sure they are completely removed. One Product Owner ranked question 10 as most needed. According to this participant it would help reduce source code, and with that the number of bugs, which would in turn reduce the cost of maintenance.

7.3.3 Are there pieces of information about the system(s) you work with that you would like to have but are not present in the list above? Why would you like to have these pieces of information?

One of the DevOps Engineers was interested in the role of classes in the system: whether a class is an "Entity, ValueObject, Repository, Service, Helper, Action, Event etc." Based on characteristics such as method names or interaction with others, it should be possible to group them. This would be important because it might give insight into the importance of the class or package in the system.

According to the other DevOps Engineer the information model could together with the dependencies help define the complexity of the system. Applications that are maintained for one business object without complex relations would in principle be easier to maintain than applications that are used for multiple business objects and multiple (complex) relations. The complexity would increase when these relations are additionally maintained by a third party. Besides that, the issue list would be useful, including priorities. A system with low code quality without issues would be a good system from a business perspective.

One Product Owner commented that he/she would like to know the unit test coverage of the system he/she works with. Another Product Owner wanted to know the interdependence between losely coupled system parts / micro services.

One of the Software Architects that responded commented live monitoring insights would be a good addition, because then the stakeholders can act proactively when performance degrades. Another architect would like to see the definition of API's, used frameworks, and size of traffic i.e. how much data is send/received through the application. The fourth architect that responded would like more information on memory usage of instances, and the instance creation / destruction rate. The last architect that responded would like to see code quality numbers.

Two of the Software Developers that responded filled out ideas for additional information. The first noted that he would like to have information on bottlenecks, to see if performance and responsiveness can be improved. The second developer noted that a logical view of the software architecture would be useful. It would show the modules and from there enable drilling down to details. He/she would like to see a representation of that in a graph model. According to this developer code quality can already be measured by the Software Improvement Group (SIG). He/she claims that a lot of the things in the list can also be delivered by them, so the solution should be an add on to that.

7.3.4 Do you have any feedback or suggestions?

Two respondents added another suggestion as an answer to this question. One said that the research should be expanded to functional interdependence. The second one said it would be useful to have the ability to look at figures of a specific slow request or one failing.

7.4 Conclusions

The participants mention several static code analysers that they use to analyse design flaws and poor code quality. They note that tools already exist that can be used to identify dead code design- and run-time. Additionally, they claim that there is a tool that can show in which methods exceptions originate. This makes questions 4, 7 and 8 less important to answer.

Several reasons were given for ranking the questions in the list as such. Most of the reasons provided are related to performance such as: identify where performance efforts should be focused, pinpoint which parts of the code slow down user requests, and limit the number of bottlenecks. The second most mentioned reasons are related to improving understanding of the system. This is in line with the two questions that are ranked as most important by the respondents which are numbered as 1 and 2 in Table 8.

Information that is missing in the provided list is: the importance of a class or package in the system, information/data flow, the issue list including priorities, unit test coverage, interdependence between loosely coupled system parts (micro services), the definition of API's and used frameworks, memory usage of instances, and instance creation and destruction rate.

In conclusion, the survey results show that the most important questions that the stakeholders have about the system, which cannot already be answered by the currently available tools, are questions 1, 2, 3, and 6. Based on the comments in the survey, these questions can be reformulated to:

1. Which part(s) of the system contain calls that take relatively long to execute?
2. The calls/methods that take a long time: how often are those called?
3. The calls/methods that take a long time: which path do those follow through the system?
4. Which classes are called most often (and therefore important)?
5. What are the run-time dependencies of the system on (third-party) libraries?

8 Idiom Selection

This chapter describes the selection of idioms to include in the dashboard design. Since the selection and design of visualisation idioms is dependent on the task(s) at hand and the data available [18], sections 8.1 and 8.2 first provide the key characteristics of both. After that, the selection of idioms based on this information is described. By doing so, this chapter aims to answer sub-research question 4:

SQ 4. Which visualisation(s) can answer the questions identified in SQ 2?

8.1 Task identification

The questions identified through the analyses in chapters 5, 6, and 7 result in a number of tasks that a user should be able to perform with the dashboard. To answer the first question, the user should be able to obtain an overview of the system's structure given a specific scenario, and identify through which parts calls are made that take relatively long to execute. To do this, the user should be able to compare different execution times of calls to evaluate which ones are considered as "relatively long".

Once the user has identified these calls, it is important to know how often those calls are made. If a call is only made once a month, it might not be worth the effort to optimise the source code that controls this call. If the user has decided that a call is important to optimise, it would be interesting to know which path these 'long' calls take through the system. This could aid in understanding whether the call itself takes longer to execute, or it is due to the subsequent call.

Next, it would be interesting to know which classes are called often in general, because this indicates that they are important within the system. Last, when a call takes 'longer' to execute or is in any other way interesting, it is important to know whether the call is part of a (third-party) library. The user should therefore be able to quickly make a distinction between internal and external class objects i.e. class objects that the user can change or not.

8.2 Relating tasks to data

The identified tasks can be broken down into two key components: questions about the runtime structure of the software and questions about performance metrics in relation to this structure. Therefore, the dashboard should include a visualisation of the overall structure of the software as well as ways to visualise metrics in relation to this structure.

The structure to be visualised can be viewed as a hierarchical network of interactions between class objects that call each other. The Java programming language has an inherent hierarchy within the used software elements: programs are organised as sets of packages, which have a hierarchical structure, and members that are class and interface typed objects [73]. An example of a class that calls a method of another class is provided in Figures 23 and 24. The class named "CallerClass" creates an instance of the class "CalleeClass" and calls the method "foobar()" that is present in the called class. Because of these hierarchical interactions, we studied idioms that could potentially visualise structure, interaction and flow through a system in a scalable way.

Figure 23: *Example of a class (that could be called).*

```
package example.callee;

public class CalleeClass {
        public void foobar() {
                System.out.println("Hello World");
        }
}
```

Figure 24: *Example of a caller class calling a method in the callee class.*

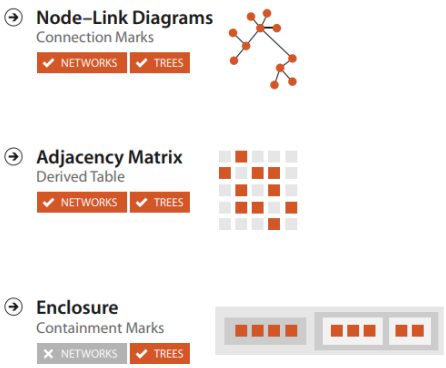
```
package example.caller;
import example.callee;

public class CallerClass {
    public void someMethod() {
        CalleeClass calleeInstance = new CalleeClass();
        calleeInstance.foobar();
    }
}
```

8.3 Visualising structure

According to Munzner et. al [18], there are three design choices for arranging networks: 1) node-link diagrams, 2) adjacency matrix, and 3) enclosure, which can only be applied to tree data. These choices are illustrated in Figure 25.

An adjacency matrix is a representation in which all nodes in the network are laid out along the vertical and horizontal edges of a square region. Matrix views of networks can achieve very high information density, up to a limit of one thousand nodes and one million edges, just like cluster heat-maps and all other matrix views that use small area marks. The key weakness of matrix views however is their lack of support for investigating topological structure, since links are shown in a more indirect way than the direct connections of node-link diagrams. The study of Ghoniem et al. [74] confirms this: by comparing the effectiveness of matrix and node-link diagrams their study found that for most tasks matrix views are more effective, except when identifying multiple-link paths between nodes.

Figure 25: *Design choices for arranging networks according to Munzner et. al [18].*



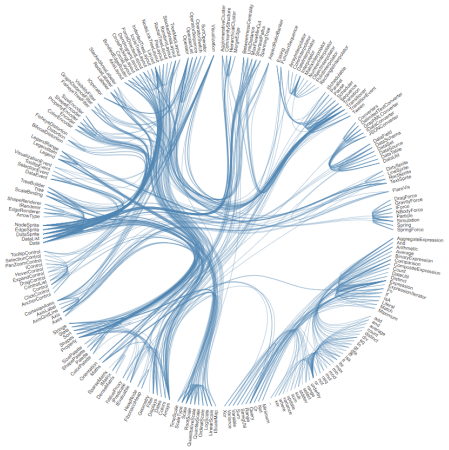
The second category of network arrangement is enclosure. Treemaps are a good example of this: hierarchical relationships are shown with containment rather than connection. All of the children of a tree node are enclosed within the area allocated to that node, and the size of nodes is mapped to some attribute of the node. These idioms are very scalable, with up to one million nodes and links [18]. However, these are also not as effective as pairwise connection marks for tasks focused on topological structure such as path tracing through the tree. Since path tracing is an important part of the tasks that our dashboard should be able to fulfil (8.1), we choose node-link diagrams as the type of idiom to visualise software structure with within the dashboard design. The connection marks in node-link diagrams support path tracing via measuring the number of discrete 'hops' of links between nodes. While it is algorithmically straightforward to design 3D layout algorithms, it is rarely an effective choice because of the many perceptual problems discussed in chapter 4.2. This is why we choose to use a two dimensional representation.

One of the most used idioms for node-link network layouts is force-directed placement of the marks. This type of placement has many variants though one common weakness: the layout quickly degenerates into a hairball of visual clutter with even a few hundred nodes. Therefore, more recent approaches to scalable network drawing are multi-level network idioms, where the original network is augmented with a derived cluster hierarchy to form a compound network. This can scale to 1,000-10,000 nodes and 1,000 to 10,000 links, with a node/link density of L < 4N [18].

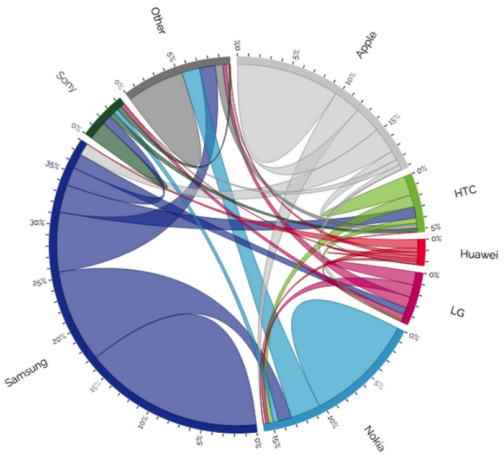
8.3.1 Radial charts

Two examples of the use of clustering to make node-link diagrams less cluttered are shown in Figure 26. Both figures use a radial layout. The first figure, Figure 26a, utilises hierarchical edge bundling to reduce the clutter of links and groups nodes by a certain category, whereas the second figure, Figure 26b, does so by aggregating links and nodes according to their source, target and direction. Though the edge bundling used in Figure 26a shows a clever way of reducing link clutter, a study by Draper et al. [75] found that cartesian visualisations tend to outperform their radial counterparts, especially with respect to answer times. They are mostly suitable for tasks that focus on a particular dimension, rather than several. Moreover, the visualisations are less scalable in the number of nodes that can be used, for the maximum number of nodes is limited to the size of the circumference of the circle. Because we are looking for a network diagram that is scalable and effective for multidimensional data, this type of layout is not preferable.

Figure 26: *Radial approaches.*



(a) *Radial chart with hierarchical edge bundling [76].*

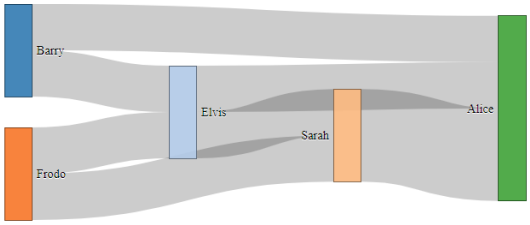


(b) *Chord diagram [77].*

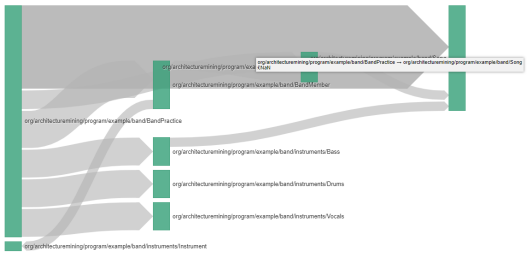
8.3.2 Sankey diagrams

Another considered option, which aggregates links and nodes according to link's shared sources and targets, is the sankey diagram. It was originally designed to analyse flow in energy and material management systems and is widely used in the field of industrial ecology [78]. Its efficiency in visualising flow makes it a very interesting candidate to answer questions regarding flow in a software system. As can be seen in Figure 27, it can be very insightful when comparing categories of components, or even hierarchical, bi-directional flow as shown in Figure 27b. However, the low-level nature of our research questions ask for the representation of specific method calls within the system, not aggregated relations between packages or modules. This level of detail makes sankey diagrams less suitable for our needs.

Figure 27: *Sankey diagrams.*



(a) *Sankey diagram [79].*

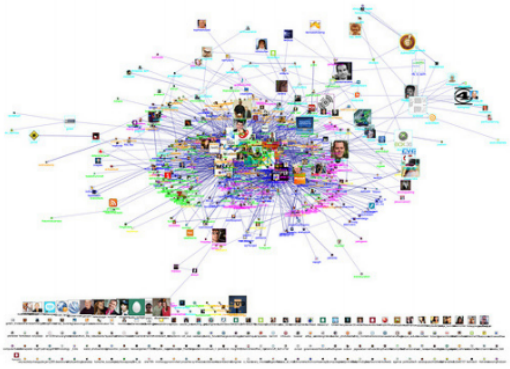


(b) *Bi-directional hierarchical sankey diagram applied to the application "BandExample" [80].*

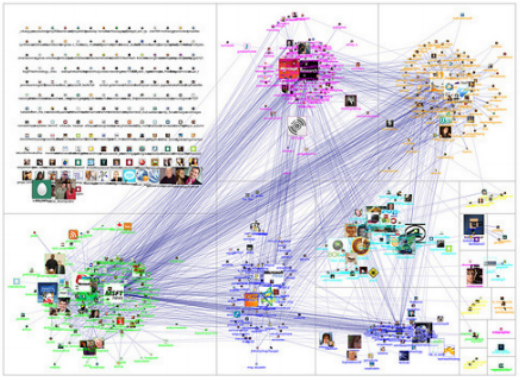
8.3.3 Combining node-link diagrams with enclosure

This brings us to our chosen solution: the Group-In-a-Box Layout for Multi-faceted Analysis, which was created for the analysis of communities. It uses the treemap space filling technique to display each graph cluster or category group within its own box, sized according to the number of nodes therein [7]. This enables us to include the inherent hierarchical nature of our data in the visualisation. Also, it offers a choice of layout algorithms for optimising the layout of the sub-graphs within each region, and the assignment of visual properties to vertices and edges within and across clusters and category groups.

Figure 28: *Group-In-a-Box Layout for Multi-faceted Analysis of communities.*



(a) *Harel-Koren (HK) fast multi-scale layout of a clustered network of Twitter users, using colour to differentiate among the vertices in different clusters. The layout produces a visualization with overlapping cluster positions [81] .*



(b) *Group-in-a-Box (GIB) layout of the same Twitter network: clusters are distributed in a treemap structure that partitions the drawing canvas based on the size of the clusters and the properties of the rendered layout. Inside each box, clusters are rendered with the HK layout" [7].*

8.4 Visualising metrics

As Tufte, the famous scientist in the field of data visualisation, said as early as 1983 [82]: "Examine the data carefully enough to know what they have to say, and then let them say it with a minimum of adornment." To keep the visualisation of metrics in relation to time as simple as possible, a bar chart is chosen. The bar chart is a well-known idiom that is easy to read. It can show the change in method duration over time, and with that highlight methods that take relatively more time than other methods. Additionally, the colours of the bars can be used to represent different characteristics of methods. When potentially having to show more than a hundred methods at the same time, the bar chart is more scalable than a line chart, which would only be able to represent methods with individual lines. This would get cluttered more easily than a bar chart does. Moreover, the bar chart could potentially enable showing concurrency, by stacking bars which represent methods that occur at the same time.

8.5 Conclusions

The identified tasks can be broken down into two key components: questions about the run-time structure of the software and questions about performance metrics in relation to this structure. Therefore, the dashboard should include a visualisation of the overall structure of the software as well as ways to visualise metrics in relation to this structure.

The structure to be visualised can be viewed as a hierarchical network of interactions between class objects that call each other. Because of these hierarchical interactions, idioms were studied that could potentially visualise structure, interaction and flow through a system in a scalable way.

According to Munzner et al. [18], network diagram designs can be broken down into three categories: 1) adjacency matrix, 2) node-link diagrams, and 3) enclosure. Since the key weakness of matrix views is their lack of support for investigating topological structure, multi-level node-link diagrams were studied. These can scale to 1,000-10,000 nodes and 1,000 to 10,000 links, with a node/link density of $L < 4N$ [18]. Of the studied idioms, the node-link diagram utilising the Group-In-a-Box Layout for Multi-faceted Analysis best suits our needs. For visualising metrics, the most clear and understandable idiom that could be used to perform the required analysis was selected: the bar chart idiom.

9 Presenting Architecture Miner

This chapter describes the chosen dashboard design, which is created to answer sub-research question 4:

SQ 4. Which visualisation(s) can answer the questions identified in SQ 2?

9.1 Solution overview

We present Architecture Miner, a web-based interactive dashboard with which architectural intelligence can be mined from large-scale Java applications. An overview of the dashboard is shown in Figure 29, of which an enlarged version is attached in Appendix A. The dashboard contains three so called idioms: a tree diagram, network diagram, and bar chart. The network diagram shows the run-time structure of the application given a specified scenario. The bar chart can change to show either: 1) the sequence and duration of calls that conform to a specified threshold of duration and number of occurrences, or 2) the number of times interactions/links occur given the same type of specified threshold.

Additionally, an options menu and a timeline are shown which can be used to control the idioms. The options menu also contains key statistics about the selected scenario, namely: the number of calls being made from class object to class object, the number of unique connections between two class objects (links), the number of unique classes on which the class objects are based (nodes), and the number of packages by which the class objects are clustered in the network diagram. This will be explained in the next sections.

Tables 9, 10, and 11 show the attributes and metrics that are visualised in the dashboard, along with the corresponding encoding and an explanation where needed.

Figure 29: *Overview of the dashboard design.*

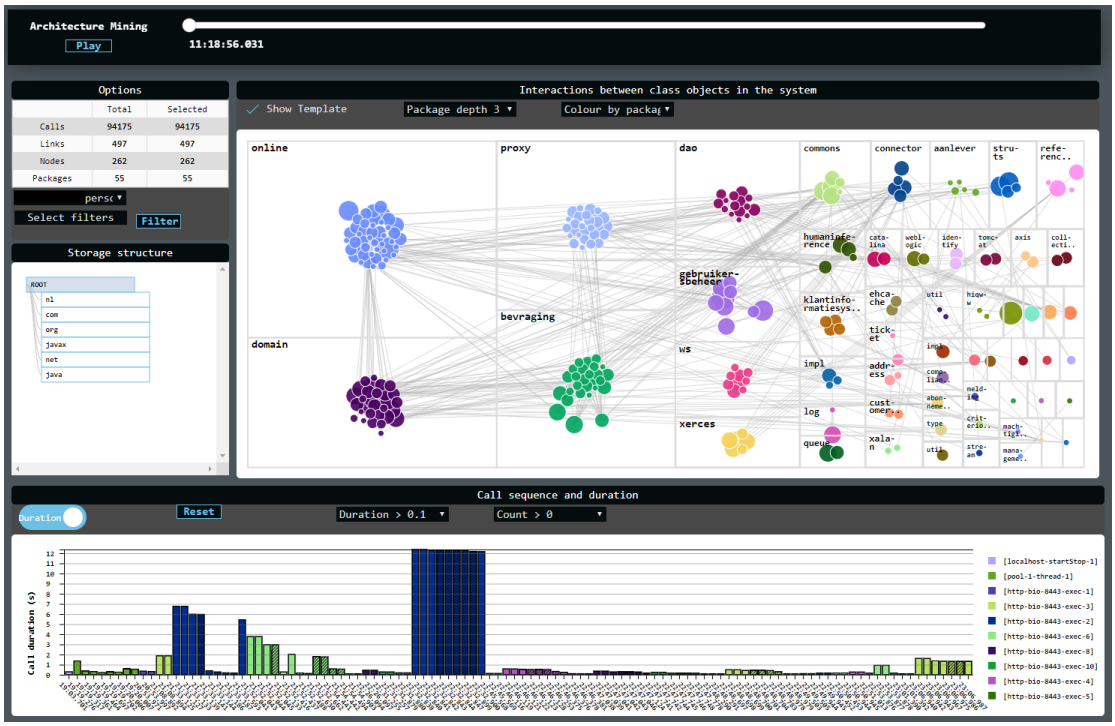


Table 9: *Encoding of the attributes and metrics represented by the tree diagram.*

Encoding	Attribute/metric	Explanation
Tree leaf: white	Package (static)	A package in the static storage structure of the code
Tree leaf: blue	Class or 'ROOT' (static)	A class in the static storage structure of the code, except for the 'ROOT' of the tree

Table 10: *Encoding of the attributes and metrics represented by the network diagram.*

Encoding	Attribute/metric	Explanation
Box	Package	
Size of box	Number of class objects within the package	The more nodes, the larger the box
Title of box	Package name	If the box is too small to display a title, it is now shown
Node	Class object	
Node position	Package	The cluster the node belongs to
Node radius	Number of instances of the corresponding class object	The more instances, the larger the radius (logarithmic scale)
Node colour	Package	Each package has its own distinct colour
Node colour: green	Negative fan in/out ratio	More outgoing than incoming connections
Node colour: red	Positive fan in/out ratio	More incoming than outgoing connections
Node colour: orange	Equal fan in/out ratio	Equal amount of incoming and outgoing connections
Node colour: blue	External class object	The class object is part of a third-party library
Link	Interaction between two class objects	Can be bi-directional
Link colour: blue	External interaction	The interaction is with a class object from a third-party library
Link colour: red scale	Sum of duration of calls over that link	The redder a link is, the longer the calls that are made over that link take in total (sum of duration)

Table 11: *Encoding of the attributes and metrics represented by the bar chart*

Encoding	Attribute/metric	Explanation
Bar	Call	Call made from one class object to another
Bar colour	Thread on which the call occurred	
Bar pattern	Sub-call	The call is also a sub-call of another call
X-axis bar chart	Start time of call	
Y-axis bar chart	Duration or count of call	

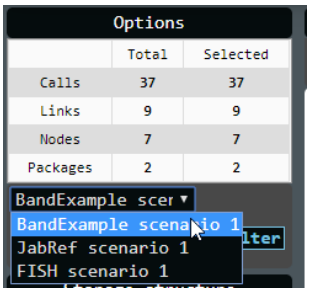
9.2 Interaction

In this section, all possible types of user interactions and their result are described. The accompanying figures illustrate what the interaction looks like in the dashboard. These figures are a result of applying the dashboard to the small Java application called "BandExample".

9.2.1 Select scenario

The options menu can be used to select the dataset i.e. scenario that the user wants to visualise. This can be done using a drop down menu as shown in Figure 30.

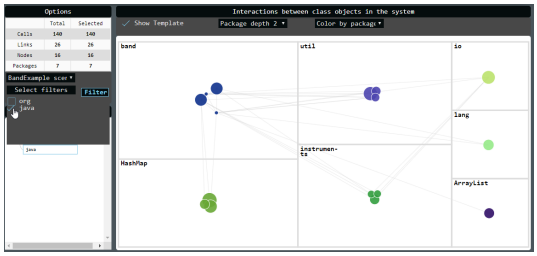
Figure 30: *Options menu: select scenario.*



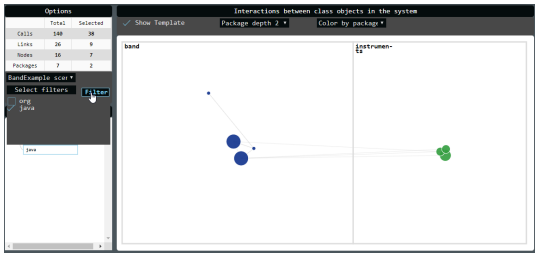
9.2.2 Filter data

Additionally, the menu contains the option to filter out class objects from the dataset on the network diagram is based. This can be done by the name of the root package of the class object. For example, all class objects which are instances of the class named "java.io.File" can be filtered out by selecting "java" in the drop down filter menu, and subsequently clicking the "filter" button. This is illustrated in Figure 31. The application statistics that are shown in the column called "selected" in the options menu and the network diagram will now be changed according to the number of calls and corresponding packages selected.

Figure 31: *Options menu: select filters.*



(a) *Before filtering.*

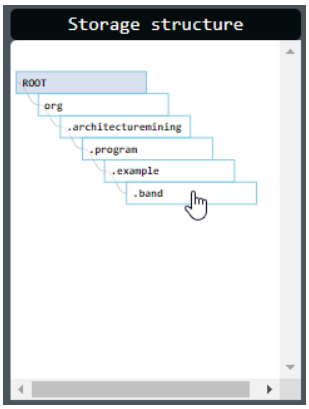


(b) *After filtering.*

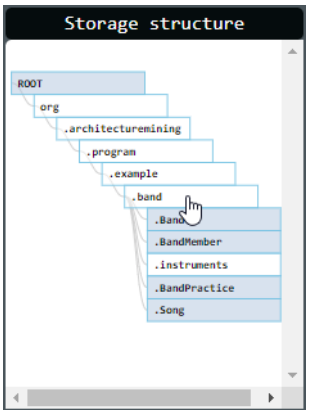
9.2.3 Browse through the storage structure

The tree diagram can be used to look through the static storage structure of the application. This can be done by clicking on a white leaf of the tree, which will make the underlying packages and classes unfold. If the selected package contains another package, this package can also be clicked and thus a dynamic way to search through the storage space is provided.

Figure 32: *Tree diagram: look through storage structure.*



(a) *Before selecting tree leaf ".band".*



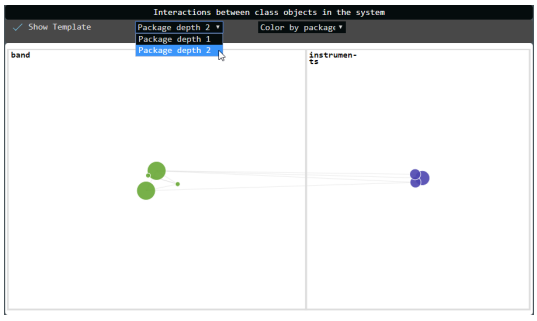
(b) *After selecting tree leaf ".band".*

9.2.4 Select package depth

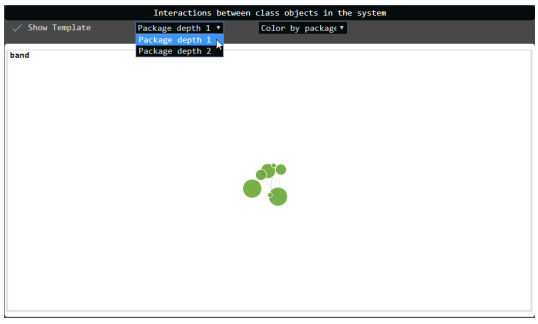
As mentioned before, the layout of the network diagram is structured according to the Group-In-a-Box layout, which uses the treemap space filling technique to display each graph cluster or category group within its own box, sized according to the number of nodes therein [7]. For example, in Figure 33a, the left box is larger than the right box because the left box contains four nodes and the right box contains three nodes.

The class objects i.e. nodes are clustered according to their package which is represented by the box in which nodes are positioned. For example, in Figure 33a, the left box represents the package called "band" and the right box represents the package called "instruments". These packages can however differ according to the chosen package depth, which the user can specify. In Figure 33a, the user has selected a package depth of 2, while in Figure 33b, the user has selected a package depth of 1, which results in both packages being coloured green. To decide which package depth to choose, the tree diagram might aid the user to look through the different package levels in the storage hierarchy.

Figure 33: *Network diagram: select package depth.*



(a) *Package depth 2.*

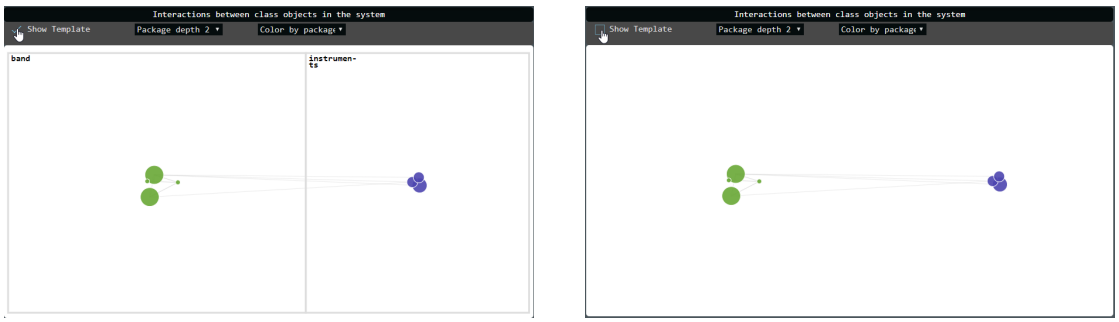


(b) *Package depth 1.*

9.2.5 Show/hide template

A template is used to show the names of packages and the sizes of the corresponding boxes to the user. Since the position of nodes already encodes to which cluster they belong, this template can be turned on and off as illustrated in Figure 34.

Figure 34: *Network diagram: show/hide template.*



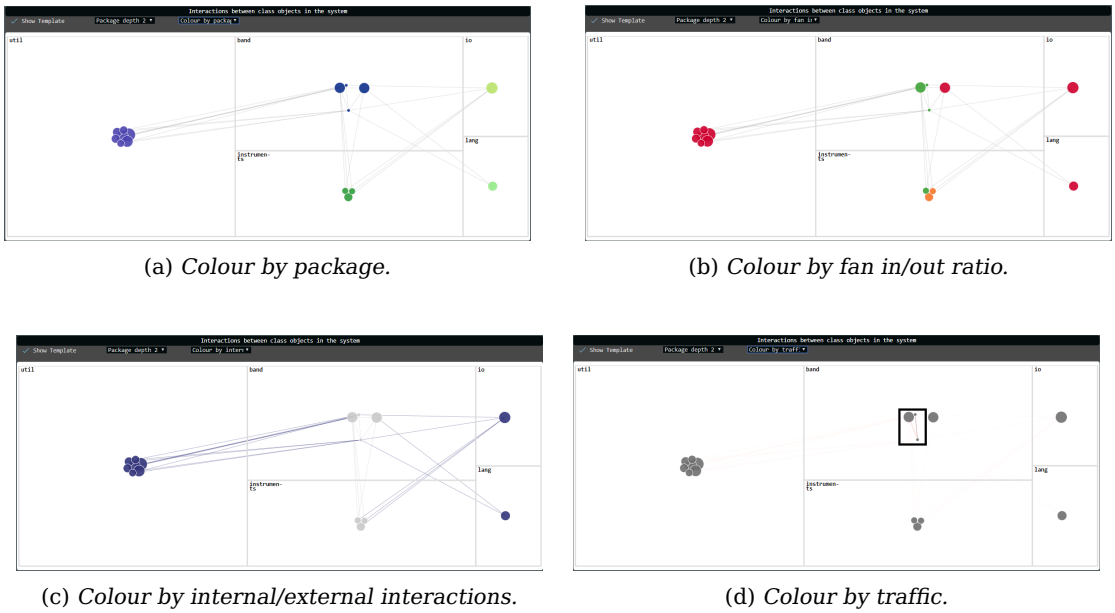
(a) *Showing the template.*

(b) *Hiding the template.*

9.2.6 Select colour overlay

The colours of nodes and links in the network diagram can be changed by choosing a colour overlay. These provide the user with additional information about the interactions shown in the network diagram. The first colour overlay, which is set as a default setting and shown in Figure 35a, is colouring the nodes of the diagram by their package. When the network diagram scales up, making it harder to gain an overview, this can provide a means to the user to see the distinctions between nodes from different packages more easily.

Figure 35: *Network diagram: types of colour overlays.*



(a) *Colour by package.*

(b) *Colour by fan in/out ratio.*

(c) *Colour by internal/external interactions.*

(d) *Colour by traffic.*

Figure 36: *Red colour scale used in the network diagram.*

Second, the user can colour nodes by the corresponding class objects' fan in/out ratio. This shows the user whether the class object has more incoming links (red), more outgoing links (green), or an equal amount of incoming and outgoing links (orange). An example of this is shown in Figure 35c.

The third overlay shows which nodes are part of or interact with a third-party library, by colouring them blue. This can be used to identify how dependent certain parts of the system are on these libraries, which is illustrated in Figure 35c. Last, links can be coloured by the amount of 'traffic' over that link. This is measured by the total duration of all calls made over that link. For example, if two calls are made over a link of respectively 0.02 and 0.5 seconds, the total duration is 0.52 seconds. The more traffic over the link, the redder the link is coloured. The used colour scale is shown in Figure 36. The scale goes from white to dark red, which is why most links will become white since their relative 'traffic' is very low. In Figure 35d, only two links stand out since they are more red.

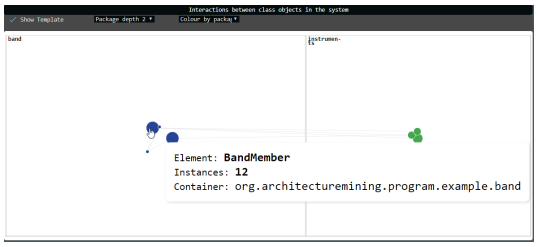
9.2.7 Hover over a node, link or box: tool-tips

Nodes, links and boxes can be hovered over to show additional information. This is illustrated in Figure 37. For a node i.e. class object the name of the class is shown, together with the number of times it is being instantiated and the container it corresponds to. In the example in Figure 37a, the class "BandMember" is instantiated 12 times and has a parent package called "org.architecturemining.program.example.band".

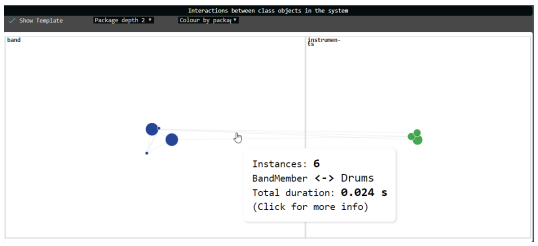
The information about a link that is shown is: the class objects between which the link represents interaction, the number of messages/calls executed over the link, and the sum of the duration of all calls over that link. In the example in Figure 37b, the link between the class objects "BandMember" and "Drums" is hovered over. The number of calls made over the link amount to six, and the total duration of those six calls added together is 0.026 seconds. The tooltip also shows "(Click for more info)", since the link can be clicked to show more info about the underlying calls in the bar chart.

When hovering over a box, the full name of the corresponding package is shown. This is especially useful when the box is too small to display a title and the title is otherwise unknown. An example of this is shown in Figure 37c, in which the box corresponding to the package "org.architecturemining.program.example.band" is hovered over.

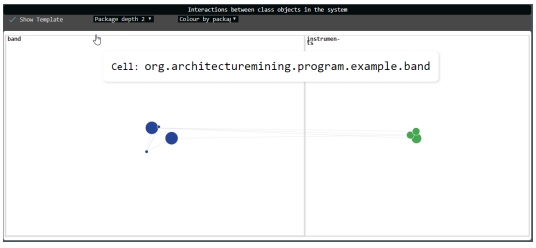
Figure 37: *Network diagram: hover over a node, link or box.*



(a) *Hovering over a node.*



(b) *Hovering over a link.*

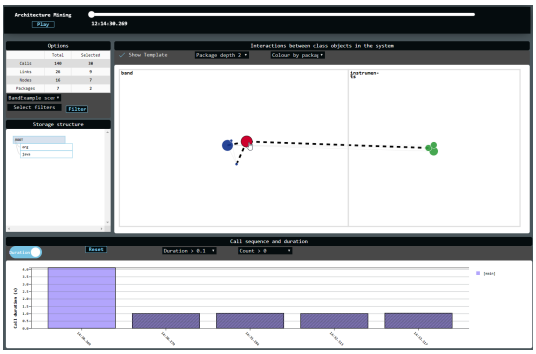


(c) *Hover over a box.*

9.2.8 Click on a node in the network diagram

When clicking on a node in the network diagram, the links connected to that node are highlighted. To make a distinction between incoming and outgoing links, incoming links are shown with a dashed line. This design is inspired by the research by Holten et. al, who study the readibility of directed-edge representations in node-link diagrams [83]. The node colour takes on the colour of its fan in/out ratio, which is red in the example in Figure 38 since there are only incoming links. If there are calls belonging to this class object that take longer than 0.1 seconds in duration, these appear in the bar chart when clicking a node.

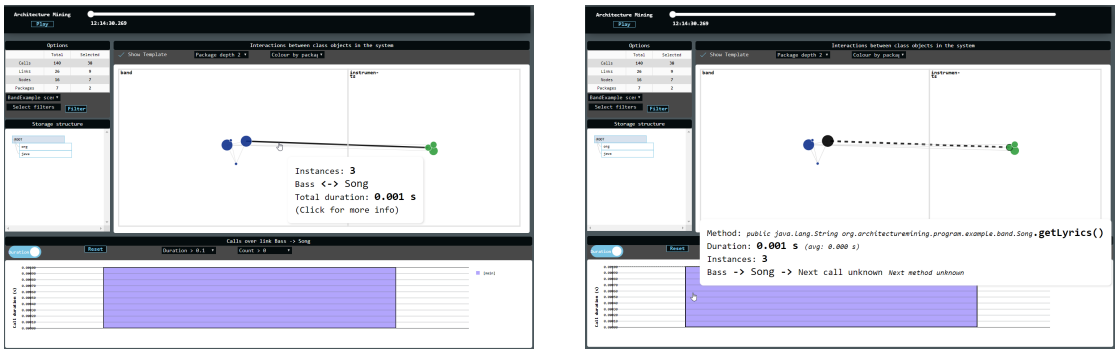
Figure 38: *Network diagram: click on a node.*



9.2.9 Click on a link in the network diagram

Clicking on a link in the network diagram makes the underlying calls appear in the bar chart. This is illustrated in Figure 39. In the example, the link between the class objects "Band" and "Song" is clicked, which reveals the underlying call to "getLyrics()".

Figure 39: *Network diagram: click on a link.*



(a) *Clicking on a link.* (b) *Hovering over the corresponding call(s).*

9.2.10 Hover over a bar: tool-tips and highlight corresponding node and links

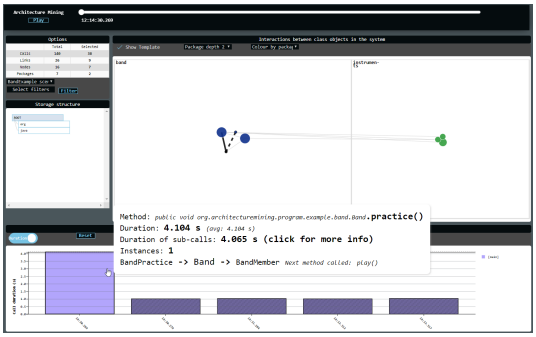
Hovering over a bar in the bar chart shows additional information about that bar, as well as its location in the network diagram. The node that corresponds to the bar is shown in the network diagram with a black fill and stroke to draw attention to it. The link over which the caller class calls the selected class is dashed, because it is an incoming link from the caller to the callee. The link that the called/selected class uses to call the next class object is shown with a regular fill. In this way, the incoming and outgoing link in respect to the hovered over class object are highlighted to the user.

For example, in Figure 40, the hovered over bar/call corresponds to the method "practice()" that is being called in the class object named "Band". This method is being called by the class object "BandPractice", which is the node connected to the dashed line in the network diagram. The next method that is called is "play()", which is called in the class object "BandMember", which corresponds to the node connected to the regularly styled link. As can be seen by the information provided in the tooltip, the measured duration of the method is 4.104 seconds. Within the scenario, it is instantiated one time.

Bars that represent calls that are also a sub-call of another call (see the definition in chapter 10.2), are shown with a print with diagonal lines over the bars. In Figure 40, all bars except the one corresponding to "practice()" have such a print, because they are all sub-calls of "practice()".

If the selected call has sub-calls, the first sub-call is seen as the next call of the call. If the call has no sub-calls, the start time of the call is used to define the next call. If the next call is unknown, this is because there is no next call, or it could not be retrieved from the data.

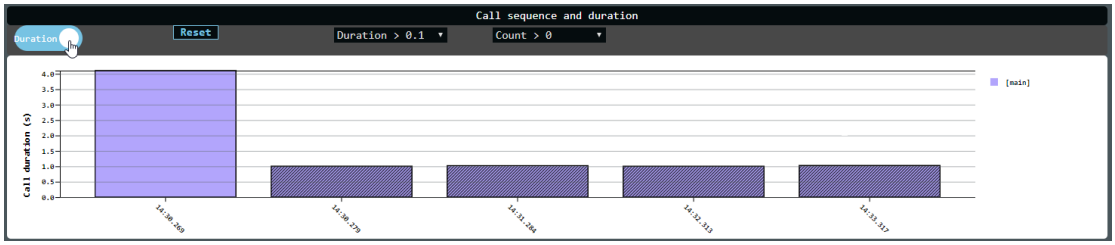
Figure 40: *Bar chart: hover over a bar.*



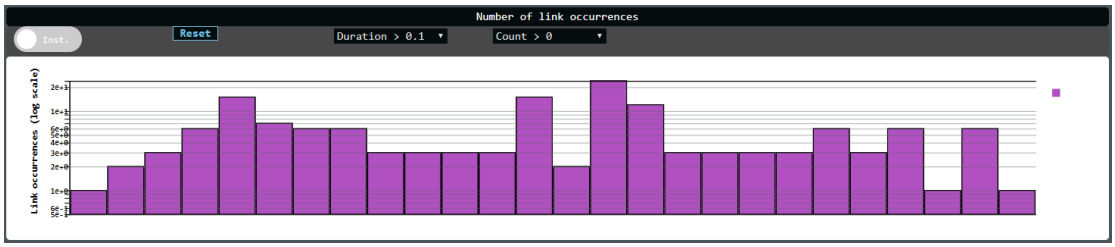
9.2.11 Toggle duration/instances

The user can toggle the bar chart view between showing call sequence and duration, and number of link occurrences. When the user has selected the "duration" view, the bars represent calls, the x-axes represent the start time of calls, the y-axes show the duration of the calls, and the colouring shows the thread on which the call was made. When the user has selected the "instances" view, the bars and x-axes represent unique links between class objects, the y-axes represent the number of occurrences of that link on a logarithmic scale, and the colouring has no meaning.

Figure 41: *Bar chart: toggle duration/instances.*



(a) *Duration view.*

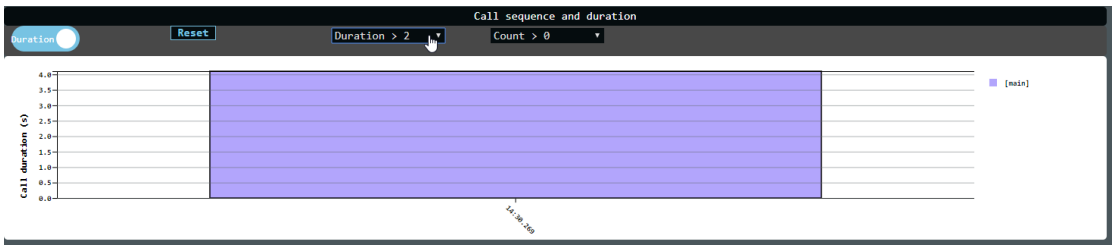


(b) *Instances view.*

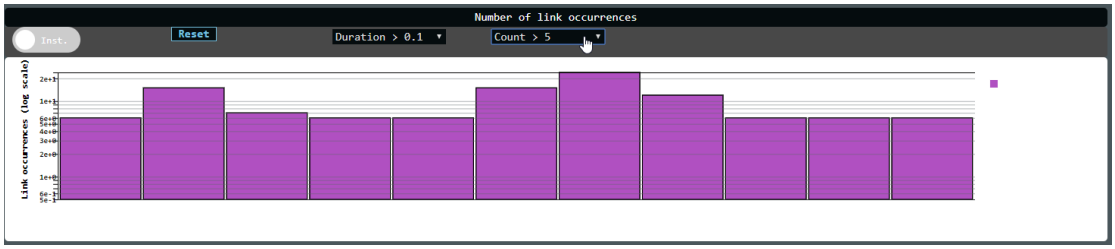
9.2.12 Set thresholds

The user can set two types of thresholds to filter the data shown in the bar chart: 1) the duration of the calls, and 2) the number of occurrences. For example, in Figure 42a, the duration view is filtered to show only calls with a duration that is longer than two seconds. In Figure 42b, the instances view is filtered to show only links that are used more than five times. A combination of filters is possible.

Figure 42: *Bar chart: defining thresholds.*



(a) *Duration view filtered by more than two seconds.*



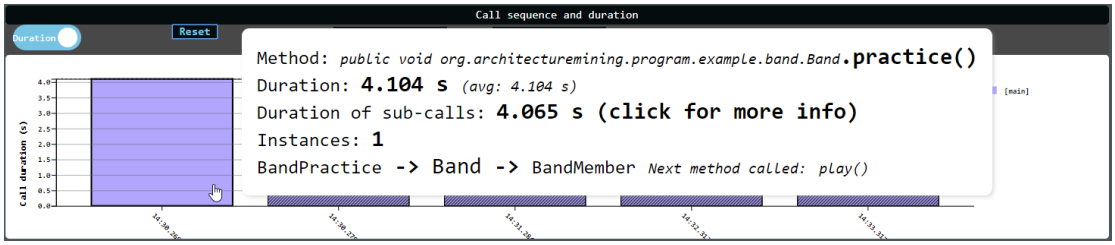
(b) *Instances view filtered by more than 5 occurences.*

9.2.13 Click a bar in the bar chart and reset it

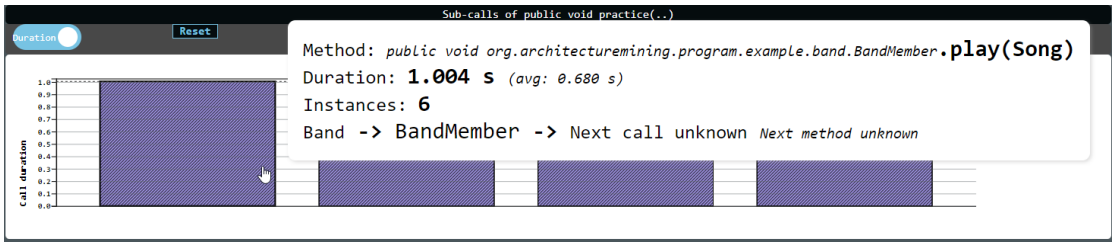
When clicking a bar in the bar chart, information about the underlying calls is shown. This can always be reset by the user to show the initial view again, by clicking the "reset" button.

In Figure 43a, a bar is clicked while the bar chart is set to the duration view. This results in showing the sub-calls of this call, in this case the sub-calls of "practice()". How sub-calls are defined is explained in chapter 10.2. As can be seen in the figure, the sub-calls of practice() are the four calls made to the "play()" method. It should be noted that in the duration view, the bar chart is filtered by calls that have a duration of longer than 0.1 seconds. It could therefore be possible that not all underlying methods are shown i.e. the methods that take shorter than 0.1 seconds are not shown. This is done to reduce clutter. As scenarios become larger, it becomes impossible to show all corresponding calls in the bar chart, which could be tens of thousands of calls.

Figure 43: *Bar chart: clicking a bar in the duration view.*



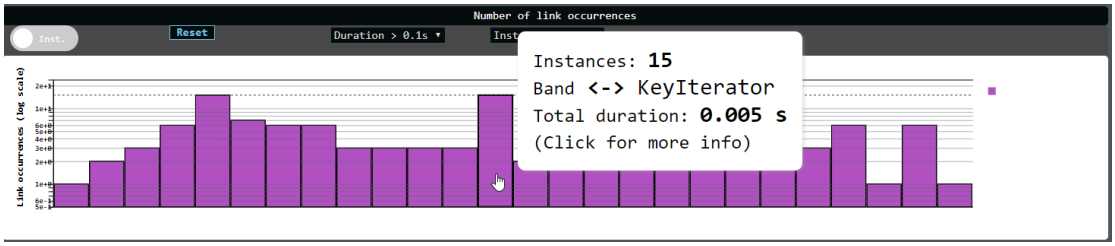
(a) *Clicking a bar in the duration view.*



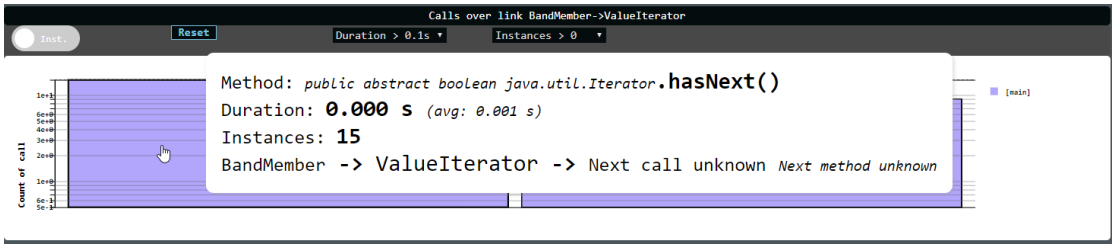
(b) *Result of clicking the bar.*

When a bar is clicked while in the instances view, the underlying calls are shown. In the example in Figure 44a, the bar corresponding to the link between the class objects "Band" and ""KeyIterator" is clicked. As a result, two bars appear in the bar chart, each representing a call made over this link. In this case, the methods "hasNext()" and "next()" are being called over the clicked bar/link.

Figure 44: *Bar chart: clicking a bar in the instances view.*



(a) *Clicking a bar in the instances view.*

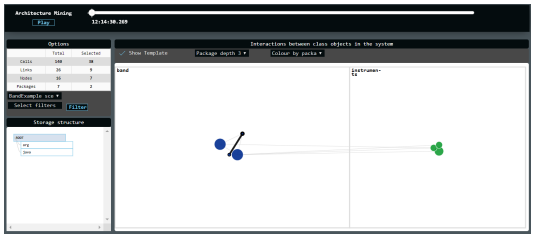


(b) *Result of clicking the bar.*

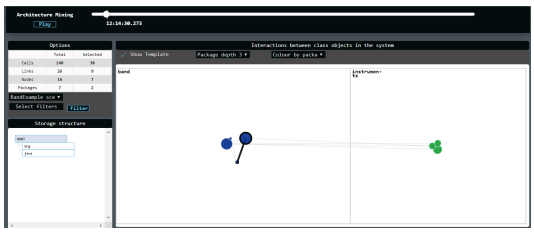
9.2.14 Drag the timeline or click play

The user can drag the timeline to see which system parts are being used at a specific time. These are highlighted with a thicker stroke width of both nodes and links. The nodes that represent class objects that are being called at that time have the thickest stroke width, followed by nodes that represent calling objects. Instead of dragging the timeline, the user can also click "play", which results in the timeline being dragged automatically.

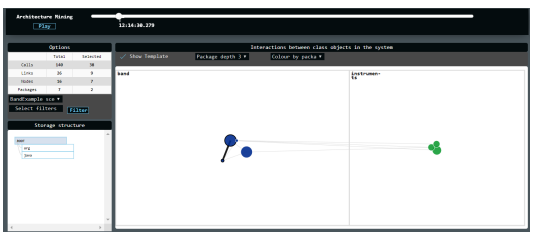
Figure 45: *Timeline controlling the network diagram layout.*



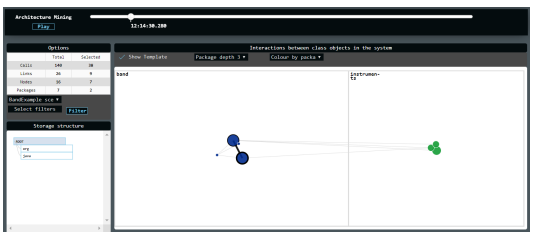
(a) *Step 1.*



(b) *Step 2.*



(c) *Step 3.*



(d) *Step 4.*

10 Dashboard Implementation

This chapter describes how the required data is collected and processed. It starts by explaining AJPOLog and the data it produces. After that, how the required information is extracted from this information is described. Next, the selection of a visualisation framework is discussed, after which the data processing and back-end design are presented.

10.1 Data collection

As described in chapter 3, the AjpoLog tool is selected to collect the data necessary to answer the research questions. Appendix A describes how to use AjpoLog to instrument Java applications. The attributes captured by this tool are [8]:

1. Timestamp
2. Name of the thread on which the call happened
3. Whether the event refers to a method entry or exit
4. Identifier (fully qualified name + object identityHashCode) of the calling object
5. Identifier of the object that contains the method being called
6. Method signature and fully qualified name of the method being called

Since De Jong (2019) [8] does not report on the influence of the tool on the duration of calls, we measure this with a small experiment. We measured the start and end time of a run of the small application called "BandExample", and repeated it ten times with and ten times without the tool. Table 12 shows the results and calculated duration of the runs with and without instrumentation with AjpoLog. It is noticeable that the runs without AjpoLog have a duration that is consistent throughout all ten runs (4.004 seconds), and the runs with AjpoLog vary consistently between 4.004, 5.005, and 6.006 seconds. The average duration of the runs with AjpoLog is 5.005 seconds. Based on these results, the tool would have an influence of 20% on the duration ((5.005-4.004)/4.004)). Although more experimentation is needed with applications of varying sizes and types to be able to generalise, it provides an indication of the influence of the tool on the instrumented system's performance.

Table 12: *Duration of ten runs of the BandExample application with and without AjpoLog.*

	Without AjpoLog			**With AjpoLog**		
Run	**Start Time**	**End Time**	**Duration**	**Start Time**	**End Time**	**Duration**
1	04:45:30.648	04:45:34.791	00:04.004	04:55:13.297	04:55:18.082	00:05.005
2	04:46:55.316	04:46:59.362	00:04.004	04:56:07.006	04:56:12.474	00:05.005
3	04:47:36.987	04:47:41.042	00:04.004	04:57:15.462	04:57:21.391	00:06.006
4	04:48:24.965	04:48:29.014	00:04.004	04:57:40.186	04:57:45.185	00:05.005
5	04:48:57.728	04:49:01.78	00:04.004	04:58:01.756	04:58:06.037	00:04.004
6	04:49:22.84	04:49:26.889	00:04.004	04:58:28.479	04:58:33.078	00:05.005
7	04:49:54.74	04:49:58.784	00:04.004	04:58:56.193	04:59:01.374	00:05.005
8	04:50:24.43	04:50:28.473	00:04.004	04:59:34.383	04:59:39.122	00:05.005
9	04:50:51.72	04:50:55.762	00:04.004	04:59:55.68	05:00:00.734	00:05.005
10	04:51:19.133	04:51:23.181	00:04.004	05:00:19.867	05:00:24.975	00:05.005

10.2 Extracting the required information

With the information collected with AJPOLog, all caller-callee relations given a specified scenario can be collected. Moreover, the data provides insight in the location of the calls by specifying the path to the called class. For example, if “org.architecturemining.program.example.band.BandMember" is being logged as a callee, it indicates that the class "BandMember" is being called, which is located in the package “org.architecturemining.program.example.band". The duration of calls can be depicted by subtracting the start time of the end time of the call. The number of times a class is called can be calculated by taking the accumulated sum of calls in which the object class is the callee. The number of times a method is called can be calculated by taking the accumulated sum of calls in which the "method signature and fully qualified name of the method being called" is equal to that method. External libraries can be defined by comparing the package and class names of the system under study by the names of the class objects found in the generated interaction logs.

As the duration of a call/method, AjpoLog measures the start and end of a call as the moment of method entry and exit. This means that the duration of all methods that that method calls is added to the duration of the method, since the methods that the method calls are executed before the method exits. Therefore, we estimate which calls are most likely sub-calls of the call/method in question, so that we can estimate the duration of individual methods. Under the assumption that all methods are synchronous i.e. blocking until they are finished, we state that: if Call2.source equals Call1.target, and interval Call2 is within interval Call1, then Call2 is a sub-call of Call1. With this information we can calculate the total duration of the sub-calls of a call, and therefore the duration of the call itself.

10.3 Visualisation framework selection

The selected visualisation framework is JavaScript:D3, which stands for Data Driven Documents. It was created by Michael Bostock as part of his PhD at the Interactive Data Lab of the University of Washington [84]. There are several reasons this framework was chosen. First of all, the authors of the framework test it against other frameworks, showing it has better performance. In their article, they describe several implemented mechanisms to optimise data management [85]. Second, since it is part of the JavaScript language, it is added to the front-end of a web application. This is a convenient type of application to be able to share the dashboard within the research community. Finally, the researcher already had (positive) experience with the framework, which made it possible to program the dashboard and try out different things in a relatively short amount of time.

10.4 Data processing

To convert the data from a LOG format to a JSON format, which is compatible with JS:D3, two Python scripts are used. The first script converts the data from a .log format to a CSV format, which is based on the script written by De Jong (2019) [8] as part of his research. The second script converts the data from a CSV format to a JSON format, and simultaneously calculates the required parameters. For example, the LOG shown in Table 46 is converted to the CSV shown in Table 47. A step-by-step instruction of how to run the scripts and use the result as an input for the visualisation is provided in Appendix A.

An illustration of the chosen JSON format can be found in Figure 57 and will now be explained. The object contains two main arrays that are called "nodes" and "links". This structure is chosen since the network diagram has nodes and links, of which the data can in this way be retrieved all at once. Links are unique connections between two classes in the system, over which calls are made. Therefore, calls are saved within the object of the link they are sent over. Sub-calls are in turn saved in the object of the call they belong to. This way the calls corresponding to links, and the sub-calls corresponding to calls can be retrieved.

Figure 46: *One method entry and exit as logged by AjpoLog [8].*

2019-08-08T12:14:30,273	[main]	Entry	org.architecturemining.program.example.band.Band	CallerPsuedoId: 1920387277	org.architecturemining.program.example.band.Song	CallerPsuedoId: 775931202	public java.lang.String org.architecturemining.program.example.band.Song.getName()
2019-08-08T12:14:30,273	[main]	Exit	org.architecturemining.program.example.band.Band	CallerPsuedoId: 1920387277	org.architecturemining.program.example.band.Song	CallerPsuedoId: 775931202	public java.lang.String org.architecturemining.program.example.band.Song.getName()

Figure 47: *One method entry and exit as logged by AjpoLog, converted to CSV format.*

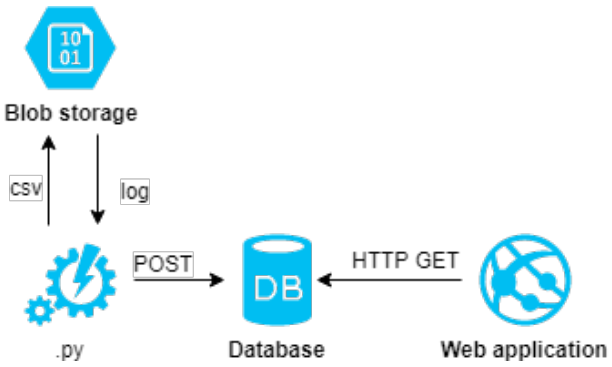
Start Date	Start Time	End Date	End Time	Thread	Caller ID	Caller	Callee ID	Callee	Message
08/08/2019	14:30,3	08/08/2019	14:30,3	[main]	CallerPsuedoId: 1920387277	org.architecturemining.program.example.band.Band	CallerPsuedoId: 775931202	org.architecturemining.program.example.band.Song	public java.lang.String org.architecturemining.program.

10.5 Back-end design

It should be noted that this format is chosen solely to suit the purpose of quickly creating a proof-of-concept: 'if it works, it works', and not to find the most efficient way of managing the data. The main purpose of this research is to test a prototype of the designed dashboard. If the visualisation should need to be more performant in the future, the back-end design shown in Figure 48 can be applied.

The log files could initially be stored in a blob storage, after which they can be processed by the Python scripts to CSV/JSON format. After that, the individual calls/objects could be posted to a database in unique key-value pairs. The web application would then be able to do GET requests to the database to retrieve the required data.

Figure 48: *Back-end design that can be applied to increase the performance of the dashboard.*



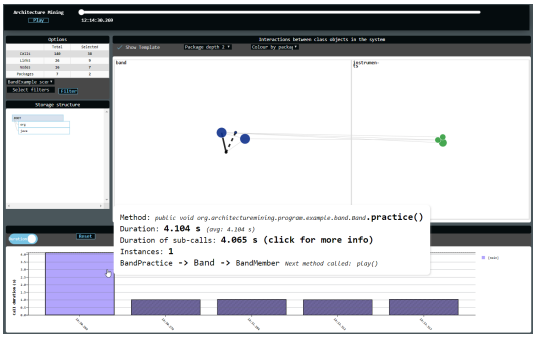
11 Dashboard Verification: Running Example

In this chapter, we will verify whether the dashboard is able to answer the research questions. We will do this by answering the questions with a small java application. The application selected to verify with is the same application that was used in chapter 9.2 to show the possible interactions with the dashboard. It is a small Java application which has 206 lines of code, 8 classes and 6 packages. The only library it uses is the Java library itself. The application has only one possible scenario, that can be executed by running the application from the command prompt.

11.1 Which part(s) of the system contain calls that take relatively longer than the other calls to execute?

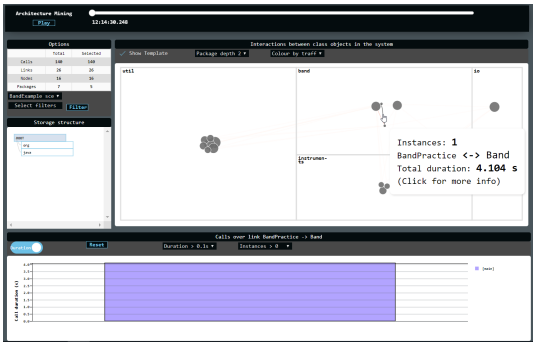
There are two ways in which this question can be answered using the dashboard. Firstly, the user can look at the bar chart and compare the bars to find the calls that take relatively longer to execute. Hovering over the bar will reveal the location of this method call that is relatively longer in duration. In this case, it could be argued that the method named "practice()" relatively takes up the most time. The tool-tip shows that this method is located in the container "org.architecturemining.program.example.band". This can also be seen by looking at the network diagram, which shows a line with a thicker stroke inside the container labelled "band".

Figure 49: *Hovering over the highest bar.*



Second, the user can switch to the "colour by traffic" view to identify the links that have the most dark red colour. Once the user has identified the link with the most dark red colour, the user can click this link to view the underlying calls in the bar chart. In this case, the link corresponds to the call to "practice()", which is why the bar corresponding to this call shows up in the bar chart.

Figure 50: *Clicking the most dark red link in the network diagram.*



11.2 The calls that take a long time: how often are those called?

Viewing how often the calls that take a long time are made can be done by hovering over the calls which reveals the tool-tip. Here a field named "instances" shows how many times that call is made during the selected scenario. In the case of the "practice()" method, this is only one time as can be seen in Figure 49.

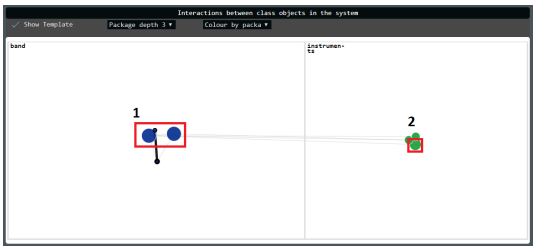
11.3 The calls that take a long time: which path do those follow through the system?

Hovering over calls of interest also reveals the path that they take through the system. On the lower line of the tool-tip as shown in Figure 49, the name of the caller class is shown, as well as the name of the class that will be called next (if known). In this case, the caller class is named "BandPractice()" and the class in which "practice()" is called is named "Band". Next, "play()" will be called in the class "BandMember". As mentioned before, hovering over a bar also reveals this path in the network diagram, with a dashed line for the incoming link and a regular line for the outgoing link.

11.4 Which classes are called most often (and therefore important)?

The radius of the nodes is defined with a logarithmic scale according to the number of times the corresponding class object is instantiated. Although area is not a the most effective way to encode ordered attributes as can be seen in Figure 15 [18], and it is difficult to compare two nodes of roughly the same size, this can be an effective way to identify which nodes "stand out". In this case as shown in Figure 51, it is clear that the nodes corresponding to the class objects "Band" and "Song" are biggest (marked as 1), followed by "Drums" (marked as 2). To compare the relative sizes of nodes, the tool-tip can be used which shows the exact numbers (12,12 and 6).

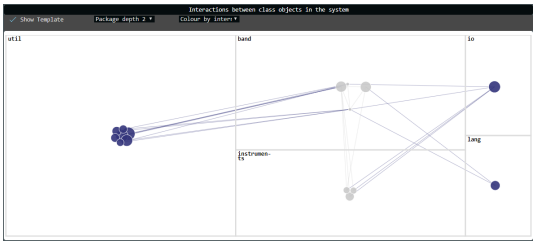
Figure 51: *Identifying the biggest nodes in the network diagram.*



11.5 What are the run-time dependencies of the system on (third-party) libraries?

To answer this question, the user can switch to the "colour by internal/external" view. From there it becomes apparent that the application uses the packages "java.util", "java.io", and "java.lang" from the Java library.

Figure 52: *Identifying the libraries that the application uses.*



12 Dashboard Validation

This chapter describes the results of validating the dashboard through interviews and a focus group, both with all four DevOps Engineers and the Software Architect. The protocols of both validation methods including the used scenario's can be found in Appendix A and A. This chapter aims at answering the sub-research questions:

SQ 4. Which visualisations can answer the questions identified in SQ 2?
SQ 5. What are the strengths and weaknesses of the created dashboard?

12.1 Participants

An overview of the system stakeholders and their experience is provided in Table 1. Since the validation took place in August 2019, four months should be added to the experience in this table. In the mean time, DevOps Engineer 3 left the team and was replaced by a new DevOps Engineer, who will be numbered DevOps Engineer 5. This new DevOps Engineer is working fulltime on the system since May, and had about one and a half year of previous working experience in a similar role. Throughout the chapter, comments will be labelled by the stakeholder who made the comments. These labels are as follows:

- DE-ME: DevOps Engineer 1, who has the most experience with the system.
- DE-MJ: DevOps Engineer 2, who has the most experience with the Java language.
- DE-SM: DevOps Engineer 4, who is also the Scrum Master.
- DE-LE: DevOps Engineer 5, who has the least experience with the system.
- SA: Software Architect.

12.2 Selected scenarios

Within the application landscape, two out of three sub-applications were instrumented with AjpoLog. The reason for this is that the third application is almost never used. A scenario is defined as follows:

Definition 25 (Scenario)**.** *A scenario is a sequence of features that trigger actions of the system and yield an observable result to the user [86].*

Definition 26 (Feature)**.** *A feature is a realised functional or non-functional requirement [87].*

Since the definitions of a scenario and a feature are not unambiguous, we let the architect of the system select a scenario for validating the tool. In this way, the definition of a scenario was made by a stakeholder of the visualisation, which would be the case if the tool would be in actual use by the DevOps team. The scenarios are included in the validation protocol in Appendix A and A.

12.3 Interviews

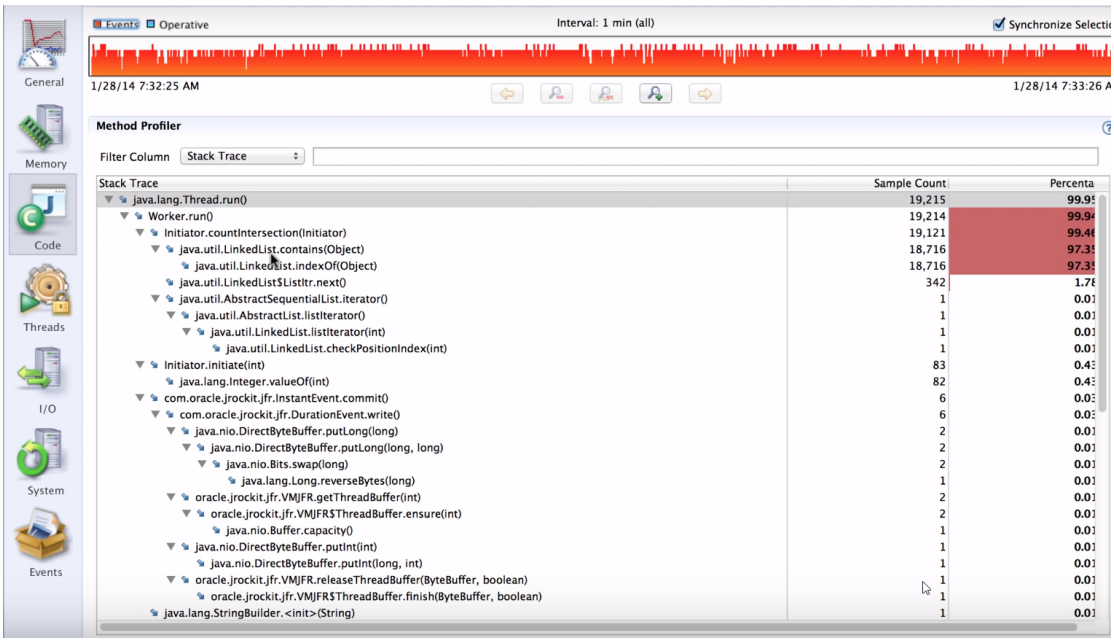
12.3.1 Answering the questions without the dashboard

To identify which parts of the system contain methods that take relatively long, all interviewees would first look at the logs produced by the monitoring systems, which are already in place in the production environment of the application. Here the total time of requests can be depicted, along with the order of classes being called, which include timestamps. The information that is missing to answer the question is the execution time of methods.

One of the interviewees (DE-MJ) would run part of the code locally, because "that is not easy in production". He would then debug the code to log information. This information would however not be complete enough to answer the question without looking into the source code.

Two of the interviewees (DE-LE, SA) note that they would additionally use a profiler to retrieve the information. Only one of them can specify which profiler he would use: Java Mission Control (SA). This profiler provides a dashboard showing the Stack Trace of program executions which is shown in Figure 53. This is noteworthy, since during the interviews held to identify the information needs of the architect, he claimed not to have this information (Table 7).

Figure 53: *Java Mission Control Call Tree [88]*



None of the four DevOps Engineers can answer the question about how often methods are being called. One of them replies that you can speculate about it by looking into the code in debug mode (DE-SM). Another notes that it can be speculated about from the logging, but only if the line of logging is unique (DE-ME). Another replies that he would add logging to the test build, but does not specify how he would approach this (DE-LE). The Software Architect would again use JMC to view the number of times that a method is used.

To identify which classes are called most often within a certain scenario, one of the DevOps Engineers would count the times the class's name occurs in the log files (DE-ME). Two others reply that they have no way of finding this out (DE-MJ and DE-SM). The last DevOps Engineer would put logging in the constructor of the class, but admits that this is a difficult task (DE-LE). The Software Architect would again look into it using JMC.

None of the interviewees are able to depict the specific path calls take through the system, with the tools they currently have available. One of the interviewees (ME-DE) replies that this can be speculated about by looking at the order of the lines of log together with timestamps and request IDs, and then into the code. This would however take a long time and it cannot be specified with certainty. Two of the interviewees (DE-SM, DE-MJ) would try to depict it by using debug mode in their development environment, and looking at timestamps in the logging. This information would however not be complete, and it takes a lot of time because break lines have to be set per individual line of code and the log files generated are very large. The architect notes that in JMC the precise order of calls does not become clear, so the source code is needed to depict that.

The fifth question can also not be depicted with the tools the interviewees currently have. The architect notes that JMC does not make a distinction between internal and external libaries. The others reply that they can only depict the static dependencies by looking into the pom.xml file of the project (DE-SM and DE-LE), look into the information that their IDE provides (DE-SM and DE-LE), or simply look into the code to see which imports each class uses (DE-MJ).

12.3.2 Answering the questions with the dashboard

The interviewees could answer all questions using the dashboard. One of the interviewees even asked: "Why are you asking me these questions, while what you have made is exactly customised to answering these questions?" When asked directly, the interviewees also replied that they could formulate a complete answer to all questions.

Nevertheless, the interviewees did have ideas for improvement that could make it easier to answer the questions. Regarding the first question: "which part(s) of the system contain calls that take relatively longer than the other calls to execute", the architect noted that it is difficult to depict to which call the shown sub-calls belong. Calls that are "deeper" in the call hierarchy are more interesting, and with the current visualisation this is hard to depict. Also, one of the DevOps Engineers (DE-MJ) found the threshold of calls that take longer than 0.1 seconds already quite high. He would like to see the bar chart in even more detail.

To all interviewed stakeholders, depicting the path that a call takes through the system is clear regarding the caller class and the class that the callee class calls next. However, the information about the class that the callee class will call next is sometimes missing (DE-ME and DE-LE). One of the interviewees (DE-MJ) noted that it would be useful to see not only the caller class, but also the method within this class that caused the callee class being called. Another interviewee (DE-SM) replied that she could answer the question, but finds further path tracing difficult sometimes. It would be useful to not only see the previous and next step highlighted in the network diagram, but multiple steps before and after the call is being made. Also, it would help if the layout in the network diagram would stay fixed to the selected call when a bar in the bar chart is clicked. Lastly, one of the interviewees (DE-LE) noted that relatively many class objects call methods within themselves, and he would be interested in seeing the call that led to this behaviour, and how many steps this was before the behaviour started to occur.

Only one of the interviewees (DE-LE) found that the way in which external libraries are visualised is not entirely clear. This is because the links in the view are fairly cluttered, which would make depicting the exact interactions difficult. Despite this, he was able to name all external libraries in the validated scenario.

12.3.3 Perceived usefulness

The answers of the interviewees regarded perceived new insights, speed of finding information, and tasks that the interviewees would use the dashboard for in their daily work. These are now described.

All interviewees replied that the dashboard provides them with new insights, because there is more detailed information about the calls being made. Especially the information about when and how often methods are being called, in combination with their duration. Moreover, it is possible to see which methods call other methods (DE-LE), and gain detailed insights into methods that are slow (DE-MJ) or could be potential bottlenecks (DE-LE). The software architect added: "I have never seen a visualisation with metrics in combination with structure before. This provides a better overview than other profilers do."

The stakeholders also agree that the dashboard would save them a lot of time. One even notes (DE-ME): "It would potentially spare me days of work looking into log files when issues arise, if it was running in production." Other reasons given for saving time are that the engineers would not have to figure out themselves where to look in the code, nor select scenarios to test (DE-MJ).

The interviewees name two types of use cases for the dashboard: 1) improving performance (DE-RD, DE-MG, DE-CP and DE-MJ), and 2) improving code quality and structure (SA, DE-MJ). To begin with, the architect thinks the dashboard would be useful to improve the structure of the application. Currently, he has no tools available that can visualise the run-time structure of applications. The visualisation would support identifying points for improvements, and looking at the application with an open view. Still, he thinks that performance tuning using the dashboard would be less useful, since he has profilers available that provide more insight in this and additional metrics such as CPU. He would not use the tool in his daily work, but to lift the code to a higher level. One of the DevOps Engineers agrees (DE-MJ), stating that as a developer he would use the dashboard to look into test scenarios. In this way, he would for example be able to spot redundant method calls that are being made more times than needed. He adds that he thinks the dashboard provides a useful overview.

Four out of five interviewees would use the dashboard for performance related tasks. One of them (DE-ME) even claims that if the dashboard would collect live data within the production environment, he would have it open on one of his screens 24/7 to immediately see where in the system it goes wrong when performance issues arise. He stresses that he finds the "colour by traffic" in the network diagram very useful, and would imagine links turning redder according to the severity of the issue. He adds that he finds it a good decision to make links that do not have a lot of traffic white, so the red links pop out even more. He thinks using the dashboard would be a great improvement to the way in which they optimise performance, especially because the dashboard provides so many details. He adds that their current tool, LogicMonitor, does not collect live data. Another interviewee (DE-SM) notes that she would use the the dashboard to identify where development efforts should be focused, and then look into the source code to see how it can be improved.

12.3.4 Perceived ease of use

Generally, the stakeholders found the dashboard easy to use. The words "intuitive", "clear", and "understandable" were used by all of them during the interviews to describe their interaction with the dashboard. Despite this, after diving deeper into their previously given answers, they did have some ideas to improve the dashboard.

First of all, two of the interviewees found some of the naming used for elements unclear (DE-ME, SA). Yet, the architect notes that naming conventions are difficult to get right, since it is highly dependent on personal preferences. One of the engineers (DE-ME) adds that some of the functionalities would stay unknown if not explained by the interviewer beforehand. For example, when a user does not know that clicking a link will result in showing extra information in the bar chart, there is a chance he will never find out. Two of the interviewees mention that the effect of clicking a bar in the bar chart is not very clear, as well as what a sub-call is (DE-SM and DE-LE). One of them adds that it would be nice to be able to drag over multiple bars to see information about all of them at once (DE-LE).

Also, the filter check-boxes should be designed the other way around: checking a box should mean that the information is selected, not removed. The filter is also hidden a bit, it should be stressed more (DE-LE). A legend would be very helpful to solve most of these issues, including instructions that can be unfolded and folded again when not needed. Lastly, one of the interviewees found it unexpected that the nodes where continuously moving on the screen (DE-ME).

12.3.5 Strengths of the dashboard design

At the end of the interviews, the interviewees were asked to provide a top three of strengths and weaknesses of the dashboard. This resulted in the following list of strengths:

- The dashboard provides a good overview of the inputted scenario (SA, DE-LE).
- The network diagram is very well constructed: it is simple, and clear what the size of the nodes, and links represent (DE-LE).
- The combination of metrics and structure is perceived as innovative and useful (SA).
- The dashboard is a good means to explore the software: lots of questions about the software arise when looking at the visualisation (SA). Examples of this are further discussed in section 12.4.
- A large amount of information is shown which opens up a wide range of possible applications (DE-ME).
- The dashboard provides more details than the profilers currently available to the team (SA, DE-SM).
- It is very useful to have drill-down options (SA).
- Because of the method call level of detail, including the duration of the individual method calls, it is possible to identify problems (DE-SM, DE-LE).
- The paths that calls take from A-Z are very clear (DE-SM).
- It is very clear which methods are called in which parts of the system and when (DE-SM).
- The dashboard is very reactive, interacting with it is fast and smooth. Only when loading a new dataset it takes some time (DE-ME).
- The timeline is very intuitive and useful. It gives the user a good feeling about the flow of control through the application. Especially if he or she has never worked with the application before (DE-LE).
- The bar chart works very well, as well as filtering it. Using colours to encode threads is a good choice (DE-MJ).
- The colour overlays are very useful (DE-MJ).

12.3.6 Ideas for improvement

At the end of the interviews, the interviewees were asked to provide a top three of improvements that could be made to enhance the dashboard. We intentionally kept asking for more feedback, to grasp the opportunity for generating ideas. This resulted in the following list of improvements to be made:

- Overall filtering of data should be improved to reduce the "interference" of data that is not of interest (SA). Four types of filters were mentioned: 1) filtering by thread (SA), 2) filtering on a deeper level (rather than just the root of the element’s names) (SA, DE-LE), 3) filtering by time interval (DE-MJ), and 4) filtering by type of object (SA).

- A distinction should be made between different types of objects (SA and DE-ME). Applications are made up of different layers, which are generally the presentation, business, and data access layers. Especially differentiating between value objects and entities could be very useful for architectural insight (SA).
- The steps composing a scenario should be kept to a minimum (SA).
- It would be a nice addition if the user can annotate elements in the visualisations themselves, to add their knowledge of the used frameworks to their classification (SA).
- Adding a better view of the aggregated interactions between packages (SA).
- As mentioned before, add a way to see clearer which calls belong to which sub-calls in the bar chart (SA and DE-LE), like how this is done in Call Tree visualisations (Fig 53) (SA).
- It should be clearer what is meant by duration and duration of sub-calls (DE-MJ). Also, it should be clearer what a sub-call is (DE-SM and DE-MJ).
- It would be interesting to see in one overview which calls belong to a class (SA).
- It takes some time to understand what you see, and know all the possibilities (DE-SM). Adding a legend or instruction could improve this (DE-SM and DE-LE).
- As mentioned before, clicking a bar should fix the corresponding layout in the network diagram for further inspection. In this way, the network diagram can make the path of the call more clear, instead of having to look at the tool-tip of the bar (DE-SE and DE-MJ).
- Nodes should be less overlapping to make a better distinction between them (DE-SE and DE-MJ).
- It can be difficult to compare the sizes of nodes (DE-SM, DE-ME), adding a third view to the bar chart that shows the instances per class object could solve this (DE-ME).
- Sometimes the next call is unknown which would be nice information to have (DE-ME).
- It would be a major addition to collect live data from the production environment (DE-ME).
- It would be nice to not only see the caller class but also the method in that class that calls the selected method (DE-MJ).

12.4 Focus group

12.4.1 General questions

Both questions are related to the way in which the used frameworks operate. The proxy classes should indeed always represent the same class, because the Spring framework creates an object between the methods called. Therefore, it might be a good idea to replace the proxy numbers with the class object names they represent in the data, to shift focus to the actual caller and callee. When a callee class of a call does not conform to the class in which the corresponding method is being called, this is because of inheritance. The way in which this is visualised now is however correct and should not be changed.

12.4.2 Scenario 1

Regarding application 2 being used within the scenario, the participants agree that it is logical since it is necessary for the request to retrieve the right information. When asked about the duration of the interaction between "SelfPopulatingCache" and "List-CacheService", the architect answered that this is a perfect example of a background process that happens to co-occur during the selected scenario, but has nothing to do with it. This is one of the reasons why he thinks there should be better filters on the visualisation, because interactions like these interfere with the results that actually matter.

One of the engineers found it noteworthy that the class object corresponding to the "OnExecute()" method is connected to a lot of other class objects. She thinks this is because the code is badly written which causes the many connections to occur. She also wondered why the "MenuComponent" class object is used so often (19,152 times). The architect had an explanation for this, namely that the front-end of the application checks every item shown in the menu recursively, to see if the user is allowed to view it. He thinks it is a good idea to look into the "PermissionsAdapter" class and optimise this process. Another notable result is that "gebruikersbeheer" is used a lot within the scenario. The team expects that the reason for this is that the scenario is run with admin permissions, which may cause this class object being called more often to check permissions. Also, a package called "weblogic" is used within the scenario. The team uses Tomcat as a server, so it would not make sense that a class object is named after another type of server. Finally, the external company register seems to be slow and in need of optimisation.

12.4.3 Scenario 2

The fact that the method "getListFieldLabel()" is called 152 times is not a problem, since this is part of a front-end process. It is also logical that "hiquality" is the most used package, as most of the scenario is a request to this external service.

The team did not see a lot of results in the visualisation that they found noteworthy. One of the engineers wondered why the class objects: "KlantInformatieServiceImpl", "AjaxViewRoot", "ApplicationImpl", and "TaskThread" are being called so often. The team was not able to specify what the role of the "KlantInformatieServiceImpl" object is within the scenario. They found it an interesting result to dive into with the source code.

12.4.4 Scenario 3

The team found the fact that the method "listBedrijven(List)" takes a long time to execute very interesting. After a long discussion, they agreed that this is probably the case due to the admin rights with which the scenario is executed. Since the admin is not assigned to any companies, the entire list of companies known to the application is checked, while for a regular user the list of companies would be small. The second question was already answered during the discussion of scenario 1.

The team again pointed out that there is a class object used within the scenario that they did not expect to be used. There were no other results that they found noteworthy.

12.5 Conclusions

With the tools that the DevOps team currently has available they cannot depict the specific run-time path calls take through the system, nor can they depict which of the libraries are used run-time. To depict how often methods and classes are being called, and what the execution times of method calls are, the architect of the team uses Java Mission Control. It is notable that the DevOps Engineers of the team do not use this tool. For them, retrieving this information costs a lot of time, and is not complete enough to depict the execution times of specific method calls and the times they are being executed.

The interviewees could answer all questions using the dashboard. With this result, the dashboard design answers sub-research question 4:

SQ 4. Which visualisations can answer the questions identified in SQ 2?

During the focus group, there were various insights that the visualisation provided that the team found interesting. First of all, some of the packages and corresponding class objects that appeared in the visualisation were not expected to be part of the scenario. An example of this is the package "Weblogic", which should not be present since this is a server package and the team uses a different server to run the web application on. Second, the large number of instances of several of the class objects was noteworthy. An example of this is the class "MenuComponent", which should not be instantiated so many times. Third, some objects such as "onExecute()" had a large number of connections to other objects which could indicate badly written code. Finally, some of the methods that took a long time to execute resulted in a discussion within the team as to why this was the case. For the method "listBedrijven(List)", they agreed on the potential cause of this.

The most important strengths of the dashboard as pointed out by the interviewees are: 1) the combination of metrics and structure, which would be innovative and provide a good overview, 2) the large amount of information shown which opens up a wide range of possible applications, and 3) the level of detail with which the information is shown, which is not available in the tools currently available to the team. The interviewees name two types of use cases for the dashboard: 1) improving performance (DE-RD, DE-MG, DE-CP and DE-MJ), and 2) improving code quality and structure(SA and DE-MJ). The stakeholders also agree that the dashboard would save them a lot of time. They used the words "intuitive", "clear", and "understandable" to describe their interaction with it.

Nonetheless, there is room for improvement. To be able to answer the identified questions in more detail and with less effort, the interviewees gave several ideas for improvement and additional functionality. To begin with, the distinction between calls and sub-calls should be visualised in a clearer way. To several of the interviewees, the definition of a sub-call was not clear and they were confused by what clicking a bar in the bar chart actually showed them. Next, the interviewees proposed four types of filtering mechanisms that could reduce "interference" of data that is perceived as unimportant. These are 1) filtering on a deeper level than just the root package, and filtering by: 2) thread, 3) time interval, and 4) type of object. Especially differentiating between value objects and entities could be very useful for architectural insight. Finally, the information about the path a call takes is not always complete, because the call that comes next is sometimes unknown, and the bar tool-tip does not show the method call that resulted in the selected method call. In these cases, the path a call takes is not entirely clear. To increase the usefulness of the dashboard the data collection process should be improved. Having the dashboard plugged into the production environment of the application would increase its applicability regarding performance related tasks. The ease of use of the dashboard could be improved by providing an instruction that can be unfolded and folded again when not needed. This answers sub-research question 5:

SQ 5. What are the strengths and weaknesses of the created dashboard?

13 Discussion and Limitations

13.1 Instrumentation

As mentioned in section 3.4.3, AJPOLog was the only tool we found that could collect data with the required level of detail. Though this tool is non-intrusive in the sense that the source code of the analysed application does not need to be adapted, it does have it shortcomings. The most important of its shortcomings is that it remains unknown what the overhead of the tool is, both in terms of execution speed and size. In section 10.1 we provided an indication of the influence of the tool on execution speed, yet without testing the instrumentation on multiple Java programs these results remain inconclusive. This makes the instrumentation unsuitable to run in a production environment, and, moreover, the correctness of the measured execution times of method calls unknown. This is a major limitation, especially because the created dashboard focuses on execution times of method calls to answer research questions.

13.2 External validity

This research focuses solely on the Java programming language and is therefore only applicable to systems that are written in this or similar programming languages (such as C#). The dashboard design is based on the characteristics of the inputted data, and thus will be poorly applicable when the inherent nature of the data changes. For example, a key strength of the dashboard design is the use of the Group-In-A-Box layout, which enables us to include the hierarchical package and class structure of the Java language in the visualisation. The size of boxes in the layout and the characteristic by which nodes are clustered would have to be assigned to other attributes. Still, when applying the visualisation to a system written in a different programming language, this might not be the best approach.

Another limitation is that both the dashboard design and validation are based on a small case study that was conducted in the Netherlands. This makes the usefulness and applicability of the dashboard in other contexts uncertain. Five out of twenty participants also work together on a daily basis in the same office space. This increases the possibility that participants might have influenced each other's opinions throughout the research process. To reach external validity, the dashboards should be applied to Java applications that are running in different contexts, and encompass various sizes, architectural patterns and frameworks.

This would also be necessary to draw conclusions about the scalability of the dashboard. The largest application that the dashboard has been applied to consists of 225,000 lines of code. Out of the tested scenario's, the largest number of measured calls is 94,175. When an application consists of millions of lines of code, it is likely that even a small scenario will result in more than 94,175 calls.

13.3 Effectiveness of the used techniques

Within our visualisation, we have used colour as an encoding for several attributes. While colour is the second most effective identify channel, this naturally does not take into account people who are colour blind. An estimated eight percent of men and two percent of women is known to be colour blind.

This is an example of how the effectiveness of visualisations can be highly subjective. Within our case study this also became apparent, because the participants showed to have different preferences for encoding and each gave discrepant suggestions to improve them within the dashboard.

13.4 Defining sub-calls

As explained in chapter 10.2, AjpoLog measures the start and end of a method call as the moment of method entry and exit. This means that the duration of all methods that that method calls is added to the duration of the method, since the methods that the method calls are executed before the method exits. We aim to make a distinction between calls and sub-calls under the assumption that all methods are synchronous i.e. blocking until they are finished. To accurately distinguish calls from sub-calls, data about the methods that are called within a method, as defined in the source code, should be added.

14 Conclusions

In this thesis we have presented Architecture Miner, an interactive web-based dashboard which takes software execution data (SED) as an input to mine architectural intelligence. To answer our main research question as presented in chapter 2, sub-questions were formulated that each address a part of the main research question. These will now be discussed first after which the main research question will be answered.

SQ 1. How can architectural information be extracted from a running software system? Techniques that aim at reconstructing the design or architecture of a system, such as software architecture reconstruction (SAR), mainly focus on functional aspects of a system and tend to ignore quality attributes [1]. Approaches that do take quality attributes into account emphasise the reconstruction of architectural artefacts, rather than compliance checking of quality attributes during the operational phase of the software [2]. This is why run-time SED is most suitable to evaluate and evolve software architecture.

SED can be collected by applying static and behaviour data extractors on a running system. A variety of static data extractor tools can be used to obtain the needed context data. However, of the studied dynamic data extractor tools, only one is suitable for architecture mining: AJPOLog. This is because this is the only tool studied so far that captures not only the method being called (callee), but also the calling object (caller).

SQ 2. Which questions should visualisations of software dynamics be able to answer in order to be valuable to system stakeholders? Based on the analyses in chapters 6 and 7, five questions that visualisations of software dynamics should be able to answer in order to be valuable to the case study's system stakeholders are formulated:

1. Which part(s) of the system contain calls that take relatively long to execute?
2. The (method) calls that take a long time: how often are those called?
3. The (method) calls that take a long time: which path do those take through the system?
4. Which classes are called most often (and therefore important)?
5. What are the run-time dependencies of the system on (third-party) libraries?

Based on a short literature review, we found that the answers to these questions might not only be valuable within the context of the case study. According to the study by Graduleva et al. [38], both managers and architects are interested in the composition of and relations between (sub-)systems, and most used or "problematic" components [38]. Architects are also interested in the composition of clusters of classes, relations between classes and components, and implications of old flows to new flows. Developers find correlated views of code, metrics, structure, and dependencies indispensable [62], and are interested in class and package information [38].

SQ 3. How are visualisation techniques and supporting tools currently used to represent software architecture?
Shahin, Liang, and Babar (2014) [5] systematically review and classify the visualisation

techniques and associated tools reported for software architecture, and how they have been assessed and applied. They identify four types of visualisation techniques used in the architecting process: graph-based, notation-based, matrix-based, and metaphor-based visualisation. 42 percents of the analysed studies by Shahin et. al (2014) [5] provide automatic tool support for the used visualisation techniques, 47 percent semi-automatic tool support, and 11 percent manual tool support.

In the studied articles, several characteristics of techniques and tools to consider when designing SAVs are identified: multiplicity of view [4, 56], dimensionality [4], medium [58], interactivity [56], implementation [5, 56], and data representation [56].

SQ 4. Which visualisations can answer the questions identified in SQ 2? The identified questions can be broken down into two key components: questions about the runtime structure of the software and questions about performance metrics in relation to this structure. Therefore, the dashboard includes a visualisation of the overall structure of the software as well as ways to visualise metrics in relation to this structure. The structure to be visualised can be viewed as a hierarchical network of interactions between class objects that call each other. Of the studied network idioms, the node-link diagram utilising the Group-In-a-Box Layout for Multi-faceted Analysis was chosen due to its ability to be both scalable and visualise hierarchy. For visualising metrics, the most clear and understandable idiom that could be used to perform the required analysis was selected: the bar chart idiom. The interviewees could answer all questions using the dashboard. The visualisations shown in the dashboard design can thus be used to answer the questions identified in sub-research question 2.

During the focus group, there were various insights that the visualisation provided that the team found interesting. First of all, some of the packages and corresponding class objects that appeared in the visualisation were not expected to be part of the scenario. An example of this is the package "Weblogic", which should not be present since this is a server package and the team uses a different server to run the web application on. Second, the large amount of instances of several of the class objects was noteworthy. An example of this is the class "MenuComponent", which should not be instantiated so many times. Third, some objects such as "onExecute()" had a large amount of connections to other objects which could indicate badly written code. Finally, some of the methods that took a long time to execute resulted in a discussion within the team as to why this was the case. For the method "listBedrijven(List)", they agreed on the potential cause of this.

SQ 5. What are the strengths and weaknesses of the created dashboard? The most important strengths of the dashboard as pointed out by the interviewees are: 1) the combination of metrics and structure, which would be innovative and provide a good overview, 2) the large amount of information shown which opens up a wide range of possible applications, and 3) the level of detail with which the information is shown, which is not available in the tools currently available to the team. The interviewees name two types of use cases for the dashboard: 1) improving performance, and 2) improving code quality and structure. The stakeholders also agree that the dashboard would save them a lot of time. They used the words "intuitive", "clear", and "understandable" to describe their interaction with it.

Nonetheless, to be able to answer the identified questions in more detail and with less effort, the interviewees gave several ideas for improvement and additional functionality. To begin with, the distinction between calls and sub-calls should be visualised in a clearer way. To several of the interviewees, the definition of a sub-call was not clear and they were confused by what clicking a bar in the bar chart actually showed them. Next, the interviewees proposed four types of filtering mechanisms that could reduce "interference" of data that is perceived as unimportant. These are 1) filtering on a deeper level than just the root package, and filtering by: 2) thread, 3) time interval, and 4) type of object. Especially differentiating between value objects and entities could be very useful for architectural insight. Finally, the information about the path a call takes is not always complete, because the call that comes next is sometimes unknown, and the bar tool-tip does not show the method call that resulted in the selected method call. In these cases, the path a call takes is not entirely clear. To increase the usefulness of the dashboard the data collection process should be improved. Having the dashboard plugged into the production environment of the application would increase its applicability regarding performance related tasks. The ease of use of the dashboard could be improved by providing an instruction that can be unfolded and folded again when not needed.

RQ: Which visualisation(s) of large-scale system's software dynamics are effective in providing valuable architectural information to its stakeholders?
It is not possible to capture the functional features and quality attributes of a complex system in a single view that is understandable by, and of value to all of its stakeholders [9]. For different viewpoints and perspectives in software architecture, different visualisations, formalisms, and abstractions are required. To scope this research, we have focused on architectural information that is valuable to the stakeholders of the system analysed in our case study.

Based on the validation results we can conclude that, within the context of the case study, Architecture Miner is effective in providing valuable architectural information to the system's stakeholders. We consider our results to be a good indication that the insights extracted with Architecture Miner can be valuable within multiple use cases in different contexts.

15 Future Work

15.1 Proving effectiveness

As described in chapter 4.2, there are different schools of thought regarding visualisation techniques. We have chosen our visualisation design based on findings in literature, but the selection bias of the researchers should be taken into account when drawing conclusions about the effectiveness of the used techniques. We have chosen to use multiplicity of view instead of a single view, two dimensions instead of three, a standard computer screen as a medium to display the visualisations, interaction instead of a static image, and implementation in the form of a web application. These choices all influence the way in which the user perceives the visualisation.

To prove the effectiveness of the used visualisations in a way that is externally valid, controlled experiments should be conducted. In these experiments, the participants could be given a number of tasks to perform with two different types of visualisations that encode the same attributes. The speed and accurateness in which the tasks are being executed by them can then be compared for the two different treatments, to show which one is the most effective.

15.2 Architecture Conformance Checking

An attempt to perform ACC was made using the Architecture Diagram used by the DevOps team of the case study, which is shown in Figure 21. Unfortunately, the monolithic nature of the application resulted in a package that is used by all of the modules shown in the Architecture Diagram. This makes checking whether there are architecture violations an impossible task, since the rules established with the diagram are not applied within the application.

An interesting use case for Architecture Miner would be to check the conformance of the architecture as found in the dashboard, to the architecture as intended by the system stakeholders. This could be added to the functionality of the dashboard, by enabling the user to define rules which the dashboard than automatically checks to show violations. An example of a tool that integrates this statically is HUSSACT [89]. The practices used within this tool could be used as a benchmark to implement this functionality in Architecture Miner.

15.3 Scalability

An idea to increase the scalability of Architecture Miner is to add different layers of abstraction to the network diagram. By aggregating nodes with the same path to one, levels of depth can be defined. Also, the back-end design should be improved to handle a larger amount of data. The performance of the current design starts decreasing as the number of calls increase. In chapter 33, an example of such a back-end design can be found.

As can be seen in Figure 26a, using the edge bundling technique to bundle links/edges together that have the same source and target can substantially reduce link clutter. This could improve the scalability and effectiveness of the network diagram, as well as provide a means to show the aggregated interactions between packages.

15.4 Completing the hierarchy of dynamic views

Salah and Mancoridis [90] present a hierarchy of dynamic views based on program execution traces. In this hierarchy, object- and class-interaction are the base and middle-view. The third and highest abstraction level are the feature-interaction and implementation views, which capture the inter-feature dependencies and classes that implement these features. The mapping of the interactions between class objects and the features they implement is not possible based on the data we have used. It would be a valuable addition to the research to add this abstraction. To do this, Feature Location techniques could be used. Feature Location is a SAR technique which reconstructs pieces of Software Architecture by focusing on one or more features. A feature is defined as the combination of an intension i.e. goal of the feature, an extension i.e. implementation of the feature, and the name that binds the two together [91]. Feature Location techniques aim to find the extension of a certain intension.

15.5 Adding functionality

During the validation of the dashboard, several ideas for additional functionality were generated to improve the dashboard. An extensive overview of these is provided in chapter 12.

References

[1] van der Werf, J., Schuppen, C.v., Brinkkemper, S., Jansen, S., Boon, P., van der Plas, G.: Architectural intelligence: a framework and application to e-learning, EMMSAD (2017)

[2] van der Werf, J.M.E., Verbeek, H.: Online compliance monitoring of service landscapes. In: International Conference on Business Process Management, Springer (2014) 89–95

[3] Ellis, G., Mansmann, F.: Mastering the information age solving problems with visual analytics. In: Eurographics. Volume 2. (2010) 5

[4] Carpendale, S., Ghanam, Y.: A survey paper on software architecture visualization. Technical report, University of Calgary (2008)

[5] Shahin, M., Liang, P., Babar, M.A.: A systematic review of software architecture visualization techniques. Journal of Systems and Software **94** (2014) 161–185

[6] Merino, L., Ghafari, M., Nierstrasz, O.: Towards actionable visualisation in software development. In: Software Visualization (VISSOFT), 2016 IEEE Working Conference on, IEEE (2016) 61–70

[7] Rodrigues, E.M., Milic-Frayling, N., Smith, M., Shneiderman, B., Hansen, D.: Group-in-a-box layout for multi-faceted analysis of communities. In: 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, IEEE (2011) 354–361

[8] de Jong, T.: From package to process: Dynamic software architecture reconstruction using process mining (2019)

[9] Rozanski, N., Woods, E.: Software systems architecture: working with stakeholders using viewpoints and perspectives. Pearson Education (2012)

[10] De Silva, L., Balasubramaniam, D.: Controlling software architecture erosion: A survey. Journal of Systems and Software **85**(1) (2012) 132–151

[11] de Souza, S.C.B., Anquetil, N., de Oliveira, K.M.: A study of the documentation essential to software maintenance. In: Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information, ACM (2005) 68–75

[12] Ducasse, S., Pollet, D.: Software architecture reconstruction: A process-oriented taxonomy. IEEE Transactions on Software Engineering **35**(4) (2009) 573–591

[13] Vogel, D.R., Dickson, G.W., Lehman, J.A., et al.: Persuasion and the role of visual presentation support: The UM/3M study. Management Information Systems Research Center, School of Management . . . (1986)

[14] Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer (2014)

[15] Cook, T.D., Campbell, D.T.: Quasi-experimentation: Design and analysis for field settings. Volume 3. Rand McNally Chicago (1979)

[16] Berander, P., Andrews, A.: Requirements prioritization. In: Engineering and managing software requirements. Springer (2005) 69–94

[17] Maletic, J.I., Marcus, A., Collard, M.L.: A task oriented view of software visualization. In: Visualizing Software for Understanding and Analysis, 2002. Proceedings. First International Workshop on, IEEE (2002) 32–40

[18] Munzner, T.: Visualization analysis and design. AK Peters/CRC Press (2014)

[19] Garlan, D.: Software architecture: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, ACM (2000) 91–101

[20] Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Professional (2003)

[21] Mary, S., David, G.: Software architecture: Perspectives on an emerging discipline. Prentice-Hall (1996)

[22] Conway, M.E.: How do committees invent. Datamation **14**(4) (1968) 28–31

[23] van der Schuur, H., Jansen, S., Brinkkemper, S.: Reducing maintenance effort through software operation knowledge: An eclectic empirical evaluation. In: 2011 15th European Conference on Software Maintenance and Reengineering, IEEE (2011) 201–210

[24] Van Der Aalst, W., Adriansyah, A., De Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., Van Den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: International Conference on Business Process Management, Springer (2011) 169–194

[25] Knodel, J., Popescu, D.: A comparison of static architecture compliance checking approaches. In: 2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07), IEEE (2007) 12–12

[26] Ipskamp, T.: A graph-based approach to capture software behavior in architecture (2018)

[27] van Heesch, D.: Generate documentation from source code. http://www.doxygen.nl/ Accessed: 2019-04-24.

[28] van Heesch, D.: Sonargraph product family. http://www.doxygen.nl/ Accessed: 2019-04-24.

[29] Fontana, F.A., Roveda, R., Zanoni, M.: Tool support for evaluating architectural debt of an existing system: An experience report. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM (2016) 1347–1349

[30] Chen, B., Jiang, Z.M.J.: Characterizing logging practices in java-based open source software projects–a replication study in apache software foundation. Empirical Software Engineering **22**(1) (2017) 330–374

[31] Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. IEEE Transactions on Software Engineering **35**(5) (2009) 684–702

[32] Logentries: The fastest way to analyze your log data. https://logentries.com/ Accessed: 2019-04-24.

[33] Loggly: Fast, powerful searching over massive volumes of log data. https://www.loggly.com/ Accessed: 2019-04-24.

[34] LogicMonitor: Learn how the logicmonitor platform works. https://www.logicmonitor.com/ Accessed: 2019-04-24.

[35] Ritmeester, J.: How to get logs (2018)

[36] Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess progranm. IEEE transactions on computers (9) (1979) 690–691

[37] de Jong, T.: Aspectj instrumentation tool. https://github.com/tijmendj/AJPOLog Accessed: 2019-04-24.

[38] Graduleva, A., Adibi Dahaj, M.: Visualization of software architecture based on stakeholders' requirements: Empirical investigation based on 4 industrial cases. (2017)

[39] Beyer, D., Hassan, A.E.: Evolution storyboards: Visualization of software structure dynamics. In: 14th IEEE International Conference on Program Comprehension (ICPC'06), IEEE (2006) 248–251

[40] Cornelissen, B., Zaidman, A., van Deursen, A.: A controlled experiment for program comprehension through trace visualization. IEEE Transactions on Software Engineering **37**(3) (2011) 341–355

[41] Oxford: Definition of vertex. https://en.oxforddictionaries.com/definition/vertex (2019 (URL accessed on 2019-07-05))

[42] Oxford: Definition of node. https://en.oxforddictionaries.com/definition/node (2019 (URL accessed on 2019-03-04))

[43] Oxford: Definition of edge. https://en.oxforddictionaries.com/definition/edge (2019 (URL accessed on 2019-07-05))

[44] Oxford: Definition of link. https://en.oxforddictionaries.com/definition/link (2019 (URL accessed on 2019-03-04))

[45] OMG: Unified Modeling Language (UML) Specification. (November 2007) Version 2.1.2.

[46] OMG: Systems Modeling Language (SysML) Specification. (September 2007) Version 1.0.

[47] Byelas, H., Telea, A.: Visualization of areas of interest in software architecture diagrams. In: Proceedings of the 2006 ACM symposium on Software visualization, ACM (2006) 105–114

[48] Zalewski, A., Kijas, S., Sokołowska, D.: Capturing architecture evolution with maps of architectural decisions 2.0. In: European Conference on Software Architecture, Springer (2011) 83–96

[49] Wu, H.M., Tzeng, S., Chen, C.h.: Matrix visualization. In: Handbook of data visualization. Springer (2008) 681–708

[50] Langelier, G., Sahraoui, H., Poulin, P.: Visualization-based analysis of quality for large-scale software systems. In: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, ACM (2005) 214–223

[51] De Boer, R.C., Lago, P., Telea, A., Van Vliet, H.: Ontology-driven visualization of architectural design decisions. In: Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on, IEEE (2009) 51–60

[52] Wettel, R., Lanza, M.: Visualizing software systems as cities. In: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE (2007) 92–99

[53] Balzer, M., Noack, A., Deussen, O., Lewerentz, C.: Software landscapes: Visualizing the structure of large software systems. In: IEEE TCVG. (2004)

[54] Rooimans, R.: Architecture mining with architecturecity (2017)

[55] Panas, T., Epperly, T., Quinlan, D., Saebjoernsen, A., Vuduc, R.: Comprehending software architecture using a single-view visualization. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States) (2007)

[56] Gallagher, K., Hatch, A., Munro, M.: Software architecture visualization: An evaluation framework and its application. IEEE Transactions on Software Engineering **34**(2) (2008) 260–270

[57] Cockburn, A., McKenzie, B.: Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments. In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM (2002) 203–210

[58] Merino, L., Fuchs, J., Blumenschein, M., Anslow, C., Ghafari, M., Nierstrasz, O., Behrisch, M., Keim, D.A.: On the impact of the medium in the effectiveness of 3d software visualizations. In: 2017 IEEE Working Conference on Software Visualization (VISSOFT), IEEE (2017) 11–21

[59] Card, S., Mackinlay, J., Shneiderman, B.: Information visualization. Human-computer interaction: Design issues, solutions, and applications **181** (2009)

[60] Forsell, C., Johansson, J.: An heuristic set for evaluation in information visualization. In: Proceedings of the International Conference on Advanced Visual Interfaces, ACM (2010) 199–206

[61] Mattila, A.L., Ihantola, P., Kilamo, T., Luoto, A., Nurminen, M., Väätäjä, H.: Software visualization today: Systematic literature review. In: Proceedings of the 20th International Academic Mindtrek Conference, ACM (2016) 262–271

[62] Telea, A., Voinea, L., Sassenburg, H.: Visual tools for software architecture understanding: A stakeholder perspective. IEEE software **27**(6) (2010) 46–53

[63] McNair, A., German, D.M., Weber-Jahnke, J.: Visualizing software architecture evolution using change-sets. In: Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on, IEEE (2007) 130–139

[64] Diehl, S.: Software visualization: visualizing the structure, behaviour, and evolution of software. Springer Science & Business Media (2007)

[65] Babu, K.D., Govindarajulu, P., Kumari, A.: Development of a conceptual tool for complete software architecture visualization: Darch. Int'l J. Computer Science and Network Security **9**(4) (2009) 277–286

[66] Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software architecture: Foundations, theory, and practice. 2009 (2009)

[67] Oxford: Definition of project manager in english. `https://en.oxforddictionaries.com/definition/project_manager` (2019 (URL accessed on 2019-02-15))

[68] Cleland-Huang, J., Hanmer, R.S., Supakkul, S., Mirakhorli, M.: The twin peaks of requirements and architecture. IEEE Software **30**(2) (2013) 24–29

[69] Li, Z., Liang, P., Avgeriou, P.: Application of knowledge-based approaches in software architecture: A systematic mapping study. Information and Software technology **55**(5) (2013) 777–794

[70] LaToza, T.D., Myers, B.A.: Hard-to-answer questions about code. In: Evaluation and Usability of Programming Languages and Tools, ACM (2010) 8

[71] Kibana: Your window into the elastic stack. `https://www.elastic.co/products/kibana` Accessed: 2019-05-03.

[72] Iacob, M.E., Jonkers, H., Lankhorst, M., Proper, E., Quartel, D.: Archimate 2.0 specification. (2012)

[73] Joy, B., Steele, G., Gosling, J., Bracha, G.: The java language specification (2000)

[74] Ghoniem, M., Fekete, J.D., Castagliola, P.: On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis. Information Visualization **4**(2) (2005) 114–135

[75] Draper, G.M., Livnat, Y., Riesenfeld, R.F.: A survey of radial methods for information visualization. IEEE transactions on visualization and computer graphics **15**(5) (2009) 759–776

[76] Bostock, M.: Hierarchical edge bundling. https://bl.ocks.org/mbostock/1044242 Accessed: 2019-04-24.

[77] Bremer, N.: Storytelling with chord diagram. http://bl.ocks.org/nbremer/94db779237655907b907 Accessed: 2019-04-24.

[78] Schmidt, M.: The sankey diagram in energy and material flow management: Part i: History. Journal of industrial ecology **12**(1) (2008) 82–94

[79] d3noob: Sankey diagram using a csv file with v4. https://bl.ocks.org/d3noob/06e72deea99e7b4859841f305f63ba85 Accessed: 2019-04-24.

[80] Atkinson, N.: Bi-directional hierarchical sankey diagram. http://bl.ocks.org/Neilos/584b9a5d44d5fe00f779 Accessed: 2019-04-24.

[81] Harel, D., Koren, Y.: A fast multi-scale method for drawing large graphs. In: International symposium on graph drawing, Springer (2000) 183–196

[82] Tufte, E.R.: Improving Data Display. The Visual Display of Quantitative Information, Cheshire, Conn.: Graphics Press (1983)

[83] Holten, D., Isenberg, P., Van Wijk, J.J., Fekete, J.D.: An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs. In: 2011 IEEE Pacific Visualization Symposium, IEEE (2011) 195–202

[84] : Uw interactive data lab. https://idl.cs.washington.edu/about/ Accessed: 2019-08-16.

[85] Bostock, M., Ogievetsky, V., Heer, J.: D^3 data-driven documents. IEEE transactions on visualization and computer graphics **17**(12) (2011) 2301–2309

[86] Rumbaugh, J., Jacobson, I., Booch, G.: Unified modeling language reference manual, the. Pearson Higher Education (2004)

[87] Sartipi, K., Dezhkam, N.: An amalgamated dynamic and static architecture reconstruction framework to control component interactions 259. In: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE (2007) 259–268

[88] Oracle: Java mission control. https://www.oracle.com/technetwork/java/javaseproducts/mission-control/index.html Accessed: 2019-08-21.

[89] Pruijt, L.J., Köppe, C., van der Werf, J.M., Brinkkemper, S.: Husacct home. http://husacct.github.io/HUSACCT/ Accessed: 2019-04-24.

[90] Salah, M., Mancoridis, S.: A hierarchy of dynamic software views: From object-interactions to feature-interactions. In: 20th IEEE International Conference on Software Maintenance, 2004. Proceedings., IEEE (2004) 72–81

[91] Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering. Springer (2013) 29–58

[92] Janis, I.L.: Victims of groupthink: A psychological study of foreign-policy decisions and fiascoes. (1972)

List of Definitions

List of Figures

Page

List of Tables

A Appendix 1: Interview protocol

Note: These questions are based on the article by Graduleva et. al (2017) [38]. A pilot interview was held to evaluate the protocol. After that it was decided to take this interview into account as an official result, because only the sequence of the interview questions was changed and some questions were removed. The pilot interview therefore resulted in the same information as the improved version of the interview. The results based on the interview that was initially meant to be a pilot can be found in Appendix A.1.

Thank you for making the time for this interview. The goal of this interview is to gain insight in your role within the development of the system, and which information about the system can support your work. For result processing purposes, I will record this session. Do you have any questions before we start?

To begin with, I would like to ask you some questions about your role within the team.

Part 1: Stakeholder type and tasks
1.1. What is your name?
1.2. With which system(s) do you work? Can you briefly describe them? How long do they exist?
1.3. How long have you been working with the system(s)?

Position and experience
2.1. Do you have a title for your position? What is it?
2.2. How long have you been working in a position like this one?
2.3. Do you have any experience with software design (CASE tools? UML?)
    2.3.1. If yes, how long have you been working with software design?

Tasks and system design process
3.1. Can you briefly describe one of your typical work days? What are your main tasks?
3.2. (Do you work in a team?) How many people are there in your team?
3.3. How do you divide the work within the team? What role do you usually take on?
3.4. Can you briefly explain the software design process of the system that you are working with? Where are you involved in the process?

We will now move on to the second part of the interview, which is about the sources of information about the system that you use to support your work.

Part 2: Information about the system
5.1. What information (about the system) do you use? Which sources do you consult to retrieve that information?
5.2. Can you name information that you are currently missing in your work?
    5.2.1. In an ideal world, what knowledge about the system would you have that you are now missing?
    5.2.2. Can you name the "grey" or "black" boxes of the system?

The last part of the interview is about existing visualisations of the system (see next page).

Part 3: Existing visualisation of the system

7. Are you using any visualisation of the system to support your work? If yes: go to questions 8 and 10. If no: go to question 9.

visualisation techniques

8.1. Which visualisation(s) do you use? Can you show them to me? In which context do you use the visualisation(s)? For which specific tasks and purposes? 8.1.1. How many visulizations are used? Which do you like most and why?

8.2. Does it provide the information that you need? What kind of information is it?

- 8.2.1. If no, what information is lacking? what visualisation could be used to complement the existing ones?

8.3. Do you create the visualisations yourself?

- 8.3.1. If yes, which methods (and techniques) of visualisation do you use and why?

Reasons for not using visualisation

9.1. What are the reasons for not using visualisation?

9.2. Do you think visualisation would be useful? Why?

9.3. (Do you have a mind map of the system? What does it look like?) Which information is missing that visualisation could provide you?

visualisation tools

10.1. Do you use any tools for/to support the visualisation(s)?

- 10.1.1. If yes, what kind of tools (automatic, semi-automatic, or manual)? What do you use them for and why?
- 10.1.2. If yes, do you think these tools are useful? How could the tools be improved?
- 10.1.3. If no, what are the reasons for not using tools?
- 10.1.4. If no, do you think tools would be useful? Why?

Thank you again for your time. Is there anything you would like to add or mention before we finish?

A Appendix 2: Interview Quotes

In this attachment, the quotes selected from the interview transcripts that are used as a result in this research are listed per interviewee. The interviewees validated that these quotes are an accurate representation of their answers to the interview questions. The full Dutch transcripts can be requested by e-mailing the researcher.

A.1 DevOps Engineer Quotes (13/03/2019)

A.1.1 System Under Study

1. “I think there are 40 or 50 people who use it, something like that. That is really very little in comparison.”
2. “there will be a moment at which we can really start building new things. I think that moment is coming close too, because we are already busy with improving small things”.

A.1.2 Stakeholder roles and tasks

1. “DevOps Engineer of the system”
2. “officially we are DevOps, which means that every person can do or has to do both development and operations.”
3. “Then we use bi-weekly sprints in which we, certain issues are registered in the sprint which should be completed in two weeks.”
4. “at the end of the sprint of two weeks we evaluate which issues are completed and which are not and why.”
5. “that is that program where the entire backlog is registered.”
6. “During sprint planning we do discuss it. Like this is what I do and this is what you do but not explicitly”
7. “In JIRA you can drag an issue to active or in progress and then you can mark yourself as executor.”
8. “server maintenance so to say, making sure the servers keep running”
9. “making sure applications keep running”
10. “making sure the required infrastructure is in place”
11. “all those infrastructure things and how networks work together”
12. “development is mainly writing and editing code”
13. “then I check all the logging of the servers to check whether I see something weird.”
14. “sometimes I know what it is within five minutes and sometimes I am really searching for two hours.”
15. “if that is support, that is just resolving support tickets”
16. “actually I just continue with the issues of the sprint”
17. “...that is a kind of ticketing system that they developed besides it, and there the tickets enter like there is a problem with this and that system.”

A.1.3 Sources of information about the system

1. “mainly source code and logs”
2. “for functional questions I generally just call [name of customer]”
3. “my own memory”
4. “actually [name software architect], he works on it since I think December not last year but already the year before that. So generally he knows where everything or a lot of things are.”
5. “We did not receive a lot of documentation about that. We do have the manual.”
6. “the naming is however good, so it is mostly reasonably clear what it does”
7. “there are quite a lot of things that the program logs on a wrong level.”
8. “we found out that all of that was going wrong for months without us knowing, because it was never logged somewhere that it went completely wrong”’

A.1.4 Information needs

1. “the first things I searched for then was how late it happened and what is the e-mail address of your certificate. And with those two things you could just trace at the entrance from which they enter our service what happened and finally you then end up at the error in some log.”
2. “Yes, the hardest part is currently still HI.”
3. “But I do know what happens there but only because it is closed source we cannot really look into it but we do know what it approximately does.”
4. “it is written in C, C++ one of the two. That is just a fuzzy search, which is just a search engine. It looks if you search for a specific name, what is could be.”
5. “That HI process is really our bottleneck so to speak, all our information passes through there and if it is really super busy then it cannot process that information. But what happens then is that the application previous to that, FISH or CC, that one gets no answer from HI and keeps waiting like yes I require an answer from HI. That means that that connection in the web proxy also keeps waiting and that at a certain moment the worker overflows. Then the entire system crashes.”
6. “No you cannot edit it. And that would actually be illegal because, it is really licensed.”
7. “Yes what we do really miss is a fast way to see what calls what. Because that is sometimes just a complete maze.”
8. “Then you just really have to walk through the code and see like this happens there and this happens there and that calls that piece.”
9. “what actually happens with a request from the moment it enters the system up until it is completed”
10. “we will probably do that with Kibana in the future.”
11. “So it will become easy and you can see in one look like this user enters there, that request goes there, and there it goes wrong.”
12. “How many requests go wrong per second, or per minute or per hour or whatever. How many requests do not give a status 200, say 304 or whatever, statuses that are not good.”

13. “Yes because there are quite a lot of things in the program that are logged on a wrong level. Things that I would categorize as an error because it really is, that appear in debug.”

14. “But there is a lot of information that we never received from the previous supplier, such as how much traffic we could expect.”

A.1.5 Current visualisation tools and techniques

1. “How many requests enter the system”

2. “Yes and traffic”

3. “How many workers are busy in the web proxy and therefore in our web-server”

4. “And how many are still free, because when those are all occupied no one can enter the system”

5. “If those lines turn around so to say, usually that line of the workers that are free are all at the top and the workers that are occupied are at the bottom. If that suddenly crosses then something is going on.”

6. “Actually we just need more of those dashboards or more of those graphs. In which we can see different information. Such as for example how many request go wrong, how many enter the system, how long they take.”

7. “That is all present in the logging and we could extract it if we had time to arrange that.”

8. “That is actually the only monitoring system we have at the moment”

9. “Yes that is indeed coupled to it. It looks at the servers to see whether it is still up for instance, what it also does is for example with our Tomcat is that it goes looking, that is coupled to the tomcat at a certain port. Then it goes looking to see how many requests enter and how long it takes to process those requests and those sort of things.”

10. “And moreover when that takes to long it also goes skewed again and then you hear those peeps and buzzers that you can sometimes hear in our office.”

11. “Indeed. LogicMonitor is very useful, mainly for monitoring the status of the servers. We generally know it very fast when a server is out of the air. What LM still cannot do, but what is necessary, is the monitoring of logs. LM does not show us whether the copylist job for example completely fails, or every other application that really goes out of the air. As long as the tomcat server reacts normally, LM does not notice it when the application is not working. Actually everything that is really application specific LM cannot manage, and that is really the disadvantage because that is something that we do need.”

A.2 Software Architect Quotes (05/04/2019)

A.2.1 System Under Study

1. "...and had three functional applications. And those applications are all JAVA applications that are composed of several sub-applications."
2. "There is an, application 1 is actually composed of six sub-applications."
3. "But then besides that around it it has some more supporting applications, such as the company register and the mail servers and those sort of things."
4. "Well that actually all runs in the Oracle Cloud and there we eventually use systems to do the provisioning through Terraform, Ansible and continuous devlivery server and things like that."
5. "And then for maintenance we also have a support portal for ourselves in which we register tickets."
6. "For development we have an issue management system in which we track which new functionalities have to be build, which bugs need to be solved, those sort of things."
7. "The customer started in 1995 as a foundation. The oldest application that we have taken over originates from 2006. So that one is now rebuild, before that there was another system. That is application 1, I think in 2009 or 2013 application 2 was included."
8. "...last December we already started running it in production."
9. "So we are actually working on the application for 1.5 years now."

A.2.2 Stakeholder roles and tasks

1. "Yes I was included in the project right away..."
2. "...initially I fulfilled the role of drawing up the application and the development that was needed to at least get it operational. And during the year it is actually the solution architect that was initially on it who was constructing and designing the networking systems and Cloud, that person started working less and started doing other things and now I also do a part of those tasks."
3. "Yes, I do have some other projects that I am working on. But mainly this is my main project on which I spend at least 60 percent of my time and often more."
4. "Partly it is senior developer being the, architect, yes it is in that area."
5. "It is a bit of everything what I am doing. I also do a lot of operational things so, not really one thing, it is really we are a DevOps team and everyone actually has the same role and you just have a specialisation in your role."
6. "I know the application the longest. I also know best what exactly happens and I was present when most of the important decisions were made."
7. "I've been working in IT for 20 years and with every project I have a slightly different role."
8. "Initially it is actually only that you take decisions in the software in the architecture, and that grows more towards not only software but also how the network is constructed, how the applications talk to each other and how you will do the architecture to the external world. How that has to go, which security measures you want to take, which performance measures, which availability measures. More in the direction of infrastructure..."

9. “...first I check whether there are urgent things in the system, the monitoring or whatsoever whether there is something that we in any case need to look at today.”

10. “I always try to focus on which issues there are in the sprint that are important, to then push so those will be realised.”

11. “So when nothing needs to happen there I go on to which functional things need to be build and where I think that I can help other people with the functionality to build on.”

12. “Besides that I also have a role that other days I spend more time on the present issues. That the customer likes to have, to work those out in what exactly needs to happen. Also the technological, to translate those more to the technology.”

13. “Yes in the end we now do have the others that do more of the development part. And I try to work more around that so that the development happens the right way.”

14. “But also in the trajectory that comes before that I want to be present and take care that in any case that operationally everything runs well.”

15. “So I take that responsibility, that in the end officially belong to the entire team, but because I am the most senior person I want to be sure that it happens.”

16. “Yes [name Product Owner] is mainly the person that does the direct coordination with the customer. Because that is also really the Product Owner within our team. But of course the role of analysing issues, what needs to happen and making things clear before you develop it at all. That is where I also have contact with the customer. If there needs to be communicated on a more technical level with customers or [type of participants] then I always join.”

17. “Yes I rather have that people look at the issues and based on the priorities that issues have and the issues that they think they can solve, that people just take it on themselves. Than you have that people know well what they are doing and therefore solve it quickly ór that they can choose to take something that is very unknown to them but that they would like to learn. In both cases it is often a motivating step to people to really work on it actively.”

18. “It is the case that if you have taken on an issue that you are the person that needs to take care that it will be finished. So that you take responsibility for completely taking it from a TODO to production.”

19. “Often it is actually that the customer says we want to do about this. This is a service that we want to deliver to our participants.”

20. “And actually at that moment [Product Owner name] and I sit together to see which procedure, which process is actually behind that, and which parts of that process will we have to support, with the software, with the technology.”

21. “Then next week we will first sit down with the customer to see like which process do you actually have now. And which parts do you have that we can automate.”

22. “After that we look into which software components we have that can fulfil that for them and which things that are. Then we can look into where that fits in our infrastructure, what we need to put that into our infrastructure.”

23. “Then they can work at it in the sprint as in we create a sprint planning at the beginning, to see what we have to take on when an issue is included.”

A.2.3 Sources of information about the system

1. “Well we have minimal documentation in the end even from the past trajectory about how the application is composed and where everything passes through. So in a couple of simple diagrams how the system is composed.”

2. “In production mainly a lot of logging of the servers, the statistics of what CPU and memory usage and that sort of things do. So actually the metrics that come out of that.”

3. “Besides that we have in production some more log files where we can find information about how requests go. So that we can follow a bit how requests pass through the system.”

4. “And furthermore it is more the source code that is leading in how it goes. There is not a lot of documentation about how the source code is composed.”

5. “...we actually only received the source code. In that a little bit of documentation about SOAP WSTL’s and things like that was included, but no single document about the source code itself. Even within the source code there is very little documentation. So it is purely the source code itself that tells us how things work.”

6. “And with that the Oracle Cloud does help us by telling us how much disk space we have used, this amount of network we use. Those are metrics that just come out of the environment in terms of netwerk and in terms of CPU and in terms of disk usage, those sort of things.”

7. “At this moment that is actually just our IDE on its own. And a little bit where you can see how good it is is that we use Sonarcube. But that is more a static analysis tool about how good the software is and still does not help that well in showing how everything is composed.”

8. “The static analysis for constructions that you do for bugs and some complexity rules are in there, and circular dependencies for between packages and the like. So a bit you can retrieve from that, but it detects more bad practices than that you can really get an overview out of that about how it is composed. And you actually only get that by using some plug-ins in that IDE that can show some more how the modules work together. And which classes work together, those sort of things.”

A.2.4 Information needs

1. “Well a lot of things that were really missing for us, was more how the infrastructure is configured and eventually how things work together. And what we have in the mean while recorded in pictures ourselves. That was actually initially missing when the system was transferred to us by the previous party, about how they designed the network how they divided the applications on different servers, how they talk to each other. Yes, more those lines, the high-level lines about the software were really missing. And therefore we had to create that documentation ourselves.”

2. “Exactly. So actually a lot of information about how the system is being used, how big it has to be, how much data enters, we got very little information about that from the previous supplier. And then we had to guess ourselves how we think it would be. And over time we can now adjust it now we see ourselves how it is.”

3. “But that does not really help with making your application itself insightful.”

4. “Well initially we constructed it in the most simple way in which we thought it ran at the other supplier. And now we are already looking into how we are going to develop it further towards other things. At the moment we have it running on virtual machines physically. While we want to go more towards that we use a virtual machine

from Oracle Cloud and that we run containers inside, and that we can run multiple containers on a virtual machine. While at the moment it is more that we have an application running on a machine. So we hope that we can soon shift in that more easily with where the application is running and that we are not stuck in that so much in the way we have defined it right now for ourselves. Because at the moment we say like okay this virtual machine on which this application runs talks to this other virtual machine and that will soon if we say that more like okay these two virtual machines are our docker cluster or our container-cluster and in there the containers run and on which machine that runs would not be that interesting to us anymore. And then you will look more into how many users and where that has to be shifted to then."

5. "Well I am actually someone that is quire low-level with a lot of things, so I read source code quickly and I can fairly well create a global picture for myself about how something works. I do notice that a lot of others have more difficulty with that, and then it would help to sketch more how everything works together and which parts are in the application itself."

6. "That is more for communication yes. It is mainly a tool in which you can easily browse and view on a more global level how everything is composed, that would for some people really help in developing the application."

7. "We now have three applications that are fairly monolithic, that really have a lot of source code and that is included in one application, and we are going more towards some smaller applications that are a lot less complex in composition and that those are more spread over each other."

8. "So that one also has a very clear functionality and how it is composed is often a lot more easy to understand."

9. "Well, what would be easier to begin with is depicting which things in the application are very closely coupled together, and which things you actually would like to cut from each other. Because I am sure that in this application that a lot of things do not have a clear context where they need to stay in. And that they secretly branch out to another piece. While if that would all be properly composed it should not do that. And what you mainly get with micro-services is that you lay a network layer between it, so that you cannot even do that type of branching out. And that you really have to shield that clearer in your code. And those contexts decide where those are located in the application, for that you do need some tooling to decide where that is located."

10. "Mainly it will initially be on package level. Probably even first on library level, which libraries actually talk to, because in the end every application consists of a lot of separate modules that are composed and that becomes one big application."

11. "And those small libraries you would actually want to put more separately from each other in the first place. So first on library level you would like to view what everything communicates with each other and then in that library view like okay, where does that communicate with each other, to see whether you maybe want to separate that from each other some more."

12. "Well we still learn a lot from the system, based on the source code how it works. There are enough libraries that we only received in binary form, so only closed source, of which we also do not have a clear idea how it is composed. But those closed libraries we have to replace under any circumstance the coming time for our own sources and build one from that. And that is a black box at the moment, that will have to change."

13. "And that structure because in the end the main structure that Spring composes for you is some sort of fundamental structure how the application works together and how requests go through it. So that is actually the only thing you really need maybe to understand how the application and everything works when it runs."

14. “Well in the end you do not have how the beans are connected to each other at the moment. We did not really make a visualisation of that right now. And you do have plug-ins for that in several IDE’s to show that. But most of those plug-ins work with Spring frameworks that are a lot newer and the current applications are therefore less easy to visualise in that. That is purely if you would really want to view how an application is running in production, how things are connected to each other there.”

15. “Well one of the things that we with our current application would be interesting to know is, we have received code from 2006, and you can see that over time they added a lot of things and there is really a lot of code in that that is eventually not being touched anymore. That maybe was very active but which is not the case anymore. And the maintenance of the application makes it fairly difficult to now the code that is present but with which you do not do anything in the end, that is also in the way a bit to know what does happen. So the identification of code and the removal of the code. Or it is more optimising of what a lot of people copied and pasted to remove that. That visualisation would be very useful.”

A.2.5 Current visualisation tools and techniques

1. “If I am working on architecture related things I make use of ArchiMate. That is an open-source tool in which you can fairly easily type out the different layers, for example the application layer but also more, the infrastructure layer and the, well, about the different layers in more detail.”

2. “The first picture shows what the functional applications are and how they collaborate.”

3. “Then there are some pictures that internally to the functional applications describe which libraries are used and how that is composed.”

4. “But everyone does it in a slightly different way, because [DevOps Engineer 5] who did the network designed it a bit in his own tool.”

5. “...and eventually those pictures do come together. Yes that just depends on which tool you know and what do you then use for that to make it insightful.”

6. “Not really a lot, I have a couple of pictures: which libraries are in it, and I made those last year to analyse which dependencies there were and what we were still missing and what we needed from the other supplier. That is actually the foundation of the visualisation now. But that will quickly change in, when we will really take on the code and change it. Other then that we do not really have a lot of visualisations other then what a dependency hierarchy is and things like that show from Maven.”

7. “Well you can ask Maven like tell me the libraries that these other libraries use. In the project file of every library is defined which other dependencies or which other libraries are being used by this library. And based on that you can get a reasonably large picture of who makes use of what.”

8. “...besides that it comes on a lower level regarding packages and classes. And then at the moment I did not use any other visualisation tool for that. A while ago for the previous project I did use a visualisation tool for that to identify more the circular dependencies. Yes that will probably still happen in the future.”

9. “There are some plug-ins in Eclipse that can help with that very well to show some sort of graph, a star diagram with all sort of things in which you say like this package with which other packages is that connected and which packages refer again to itself and then there, that can also very clearly show of how many packages how close they are to other packages. So how much you use of the other packages and then it often lies more close together. And based on that you can for example see how well the code is structured.”

10. “Yes because in the end when you perform that visualisation you want to see a very clean picture, some sort of tree structure, and if you do not see that then you actually know already that you have to change that.”

11. “So far yes, for this project, because that is more on a higher level and not yet very much in detail. But it does show how much is working together and which things are or are not used in the system.”

12. “Yes mainly the only visualisation that we have used is more on source code level and not that much in how the code eventually is connected in a running application. All applications make use of Spring at the moment, of the Spring framework, and the Spring framework actually has a lot of singleton beans that in the end are connected and are put together when starting the application. And that visualisation we do not have at all how that structure is connected. That you have to really read from the code yourself.”

A.3 Product Owner Quotes (05/04/2019)

A.3.1 Stakeholder roles and tasks

1. “I think it is full-time from March 2018 and I also did a part of the pre-sales so actually it is from January 2018 that I work with this system.”
2. “That company in the Netherlands was four years during which I also did software development project management…”
3. “…but as a Product Owner you are of course looking more into the trends in the market and what it is that we can do to improve. And besides that a more broad horizon to see where the future goes. Because with product development you are always behind on the facts so you have to track very carefully what the needs are and which needs will emerge. Which technologies are available or will become available, that part I try to fill in and that is still not quite, let me put it this way, it is still not quite carved in stone to put it that way. Because I of course also worked in the maintenance group for the past months with the DevOps team to support people in the process. Because the processes were not ready yet.”
4. “Check briefly whether everything still works and whether it worked well during the night.”
5. “…looking through tickets, to see whether there is something going on…”
6. “…and then the things that need to be delivered, each month I have to create a monthly report.”
7. “I usually have a meeting with the customer one time a week, one or two times a week, so you have to do some things for that and then it are a lot of questions from the customer: I need this, and how does that work, and can you give me this information.”
8. “I interfere a bit with all tasks of all people as a project manager. Actually that is not necessary, [DevOps Engineer 4/ Scrum Master name] should do that.”
9. “I try to take the time to work out the tasks or the new stories, where I do at this moment not come too that often by the way.”
10. “Well that are a lot of things, meetings, QA meetings, and then is this meeting and that meeting, people need your input.”
11. “Yes or people from the customer that need me. That just takes a lot of time and there should actually just be or come a standard answer for that. And maintain and collect that so you can in the future just refer to that like well this is the answer to your question.”
12. “I do support one day a week, also to keep understanding what is going on at the moment and to see where ops actually struggles with. And things that you know you have to do every day.”
13. “Normally the Scrum Master works 80 percent and the others a hundred. But [DevOps Engineer name] is not part of the team anymore as of next month and [DBA name] actually also not anymore.”
14. “…well I do point out the work that needs to be done, what we are going to do in the next sprint and what the future will be.”
15. “We all do a day of support, including me…”
16. “And the team divides the work between themselves that is present in the sprint. And additionally they have to take into account that, the teams are coupled two by two, so [Devops Engineer 4/Scrum Master] is coupled to [Devops Engineer 2] and [DevOps Engineer 1] to [DevOps Engineer 3], for now, that will probably change some time in the future…”

17. “...at this moment we started to do some sort of pair programming type of implementation. That you code with two people and that means that the other team, that the other two people or one of them has to ttest it and look into it.”

18. “And after each sprint a retrospective is done, what went well and what went wrong. ”

19. “...and for that they have the stand-up, so if they stand up in the morning at 10 o’clock and say like well what I will do today, what I did yesterday, what did not go well and who do I think I need, they can in that quarter of an hour that they are together, they can say I have done this yesterday, this did not go well, I have to continue that, but I need some help from you or who can help we with this and that.”

20. “...she has to coordinate that everyone, that if someone says like yes but I have no time. Look that is a bit difficult because she is also a team member, so in principle the Scrum Master does not need to be part of the team. So she has to solve all imperfections, that is the role of the Scrum Master. So if someone says something like: you know, you want me to test something for OSX but I do not have a MAC, how can I test it then? Then she has to take care there is a MAC, if that would be a requirement.”

21. “So, she has to remind people like he we are doing a stand-up, take care that you tested everything, and monitoring that so that I do actually not have to do anything with that.”

22. “At the moment we do agile development, with which we do create a general outline. For the customer we have had two innovation sessions in which we try to define the cornerstones for them: what do they find important and what needs to happen with their system or systems.”

23. “... with which we work in small parts, try to deliver small parts of functionality so there is a requirement that is very large and those are split into tasks or stories actually and those include one or more tasks and those tasks need to be so small that you can usually work them out in one sprint.”

24. “Then we try to prioritise and then every time things are added, and then we say like well what do you find important.”

25. “...when the sprint is close to being finished then we, it has to be demonstrated and then the functionality is shown, these are the tasks that we did. These are finished, these are not finished, these are demo ready, we do a demo, and this is what you are going to get. And then it is very possible that a customer says well, I would like to see it differently. Well fine, then we disapprove it and pull it to the next sprint. And if it is okay then it can go to acceptance or production.”

26. “Then usually right before or right after that you receive the first phone calls already, like he I think we get a time out, it is not working.”

27. “No that will just be with the maintenance group, because [name database administrator] cannot be a single point of failure.”

28. “And that is why it is important that the process is very clear and that you take people along in the process and say like yes, hee but look this are all the things that we still need to do and you can put them in a priority list.”

29. “Yes and in principle we decide together what is included in a sprint, that is the idea, that does not always happen but you have to decide together what will be included in a sprint like this will be in the sprint and then after two weeks then you have to be finished with your sprint. Whether things will be ready we cannot really estimate at this moment, because you have to be able to really estimate how big, how difficult

something is, or how complex something is, because you estimate with Story Points, yes that are that: that are hours or minutes, yes it are points, are it apples or pears...”

30. “Alright then you have to put Story Points in the sprint with your team...”

31. “And that point has to be spend and we assign those to an item and then you know how many Story Points are included in a sprint. And then you also know how many Story Points you completed at the end of a sprint, so then you know like well it is more work than we thought, and if the team would always work the same hours with the same composition then over time you can better estimate something and complete your sprint quicker and also that you can do more points.”

A.3.2 Sources of information about the system

1. “Well the tickets I withdraw from the ticketing system, I export those to Excel and then I hand them over.”

2. “the performance numbers come out of the logging, so at this moment [DevOps Engineer 1] extracts those for me. And then I put those in an Excel pivot table, an external data source, that will hopefully change in the future that we can just present those on a dashboard in Kibana, because that is just present in the logging so that can just be extracted.”

3. “and the requests are registered in that, specific information about that request is separate and that you do get with Excel then, with a pivot table actually all those information from that. And if I see an abnormalities then I really look into that further to see when it happened, which specific customers it where, what happened with the system then, why is it slow, or why is that, why does this one fall out, do I see a lot of errors, or do I see a lot of unauthorised enter or something like that.”

4. “...at this moment we work with a piece of software that is not documented.”

5. “And at this moment we are dependent on the knowledge of [name software architect]. And the ease with which he analysis and debugs things.”

A.3.3 Information needs

1. “...so first of all extra detail needs to be added to the log I think.”

2. “So unlocking the data, and understanding where things lie, that is something that is high on the list.”

3. “...why at specific moments specific things take a long time for example, that is purely from a SLA (service level agreement) perspective. I want to know why specific requests go slow at what kind of moment specifically.”

4. “...well this moment from two to three it was very slow. Well what happened in the system that is was that slow?”

5. “And you also do not know at this moment which code that is written, whether it is bad? What is really the bottleneck in the entire process? Because some parts we can time and say this is here now and it is now here, that went pretty fast. But you cannot see very well whether there are specific parts of code of which you can say: if we want to become efficient, where do we have to start?”

6. “...maybe you can tackle 80% of performance by hitting one piece of code somewhere because there something very impractical is being done. And that information is just not present now.”

7. “of course you want to know a lot more about the system and what happens exactly and whether it is useful to change certain things.”

8. “When a request enters that we can just follow it through the entire chain. Sometimes things happen in the system that you cannot explain. In the sense that you can of course explain it when you go through the code entirely, but even if you put it in debug mode you still do not know entirely where it now goes, if it goes. And that is something that is being improved by repeatedly changing the logging.”

9. “...that you as a matter of speaking can visualise where it went through, and which code it hit.”

10. “If you, at the moment that you say like well we have a visualisation that helps us to decide where, how the streams of data go through the system and what is exactly being hit, then you also know at a certain moment very easily whether certain parts of data, whether we will refactor certain pieces of software at all.”

11. “Because if something is never used or very little then you should maybe think about not putting effort into it, because it works, but it will take a lot of time to adjust it.”

12. “Things that are used a lot in that you would maybe want to invest. And that is very unclear at the moment of course.”

13. “...but the goal is of course in the future to, at least of application 1, what is build on a framework that is completely undocumented, because you can almost not even find it on the internet, to refactor that. And then you should know at a certain moment within the sources that you have now, which ones are really being used? Because maybe there are parts that are just death, which has never been hit by something, and those are the ones that you would actually want to amputate or remove maybe. And then you put a piece against it somewhere if that is difficult or something like that.”

14. “If you can make that very clear, that you can say like this code was being hit this many times in that month and this one this often, then you know exactly which classes you should touch.”

15. “...if you would have had a very nice architecture picture and we would have said like yes, I make this thing black but here is a connection somewhere and here is a connection and there is a connection, and that piece that circle there, black surface but you do see five lines going to it, you don’t know what it does but he. Then you at least know that those are the important things so I have to go do something with those, so that goes into that and into that, there. Because now a flow is being followed from a certain position, that goes into such a black thing, that you do find, well you know here I need a connection.”

16. “There are a lot of surprises along the way that you encounter and have to solve again.”

17. “Actually everything was an assumption, because we did not have a knowledge transfer at all.”

18. “if you say like I know how to present my software in such a way as islands with lines between them in which every class or I don’t know what is an island, then you can see how things are connected, sometimes it of course become a big spider web, but if you could zoom into that. That would of course help tremendously, because then you identify things a lot quicker.”

19. “No idea. Module I think...”

20. “...for me the highest level is the most important, because I have to look at it from a helicopter view...”

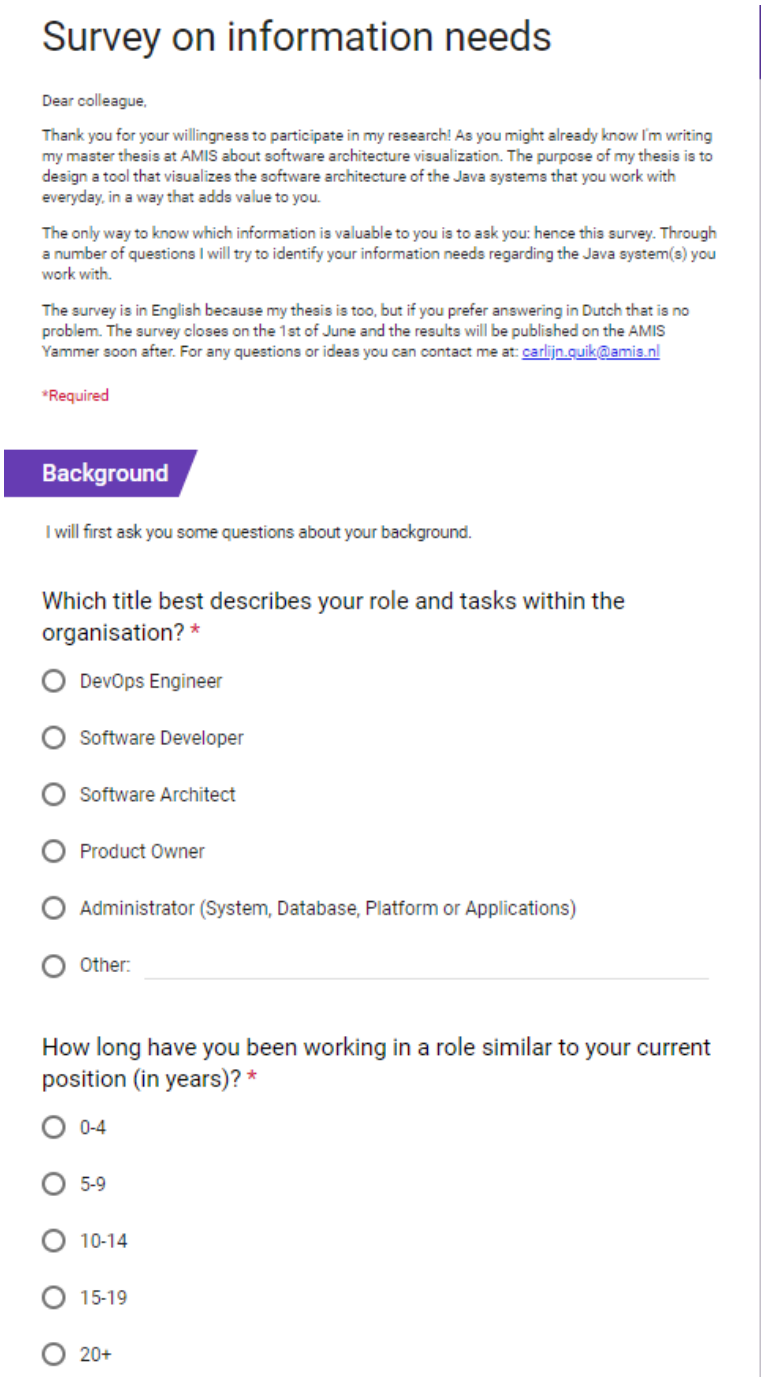
21. “And otherwise you have to puzzle and the system only says like it is buisy here...”

A.3.4 Current visualisation tools and techniques

1. “Well we never use CASE tools or UML.”
2. “...but I always use Bizagi for describing processes. That is some sort of workflow, yes, what you often see is that process description is done in Visio, but I do not find Visio very useful. In Bizagi, that is an open-source or at least freeware, but you can also get a payed version, you can more easily draw workflow type of things...”
3. “Well we of course have the LogicMonitoring like visualisations at the moment, in which we can view how the system works, or how LogicMonitoring thinks the system works, because that is still not always the same.”
4. “LogicMonitoring only says whether the servers are online and whether they are buisy and furthermore it does not say anything about reqest times or other things.”
5. “It does not say anything about the application on its own, how many requests, you can see something because there is Apache access is in there so you can see how many Apache workers are buisy...”

A Appendix 3: Survey

Figure 54: *Survey part 1/3.*



Survey on information needs

Dear colleague,

Thank you for your willingness to participate in my research! As you might already know I'm writing my master thesis at AMIS about software architecture visualization. The purpose of my thesis is to design a tool that visualizes the software architecture of the Java systems that you work with everyday, in a way that adds value to you.

The only way to know which information is valuable to you is to ask you: hence this survey. Through a number of questions I will try to identify your information needs regarding the Java system(s) you work with.

The survey is in English because my thesis is too, but if you prefer answering in Dutch that is no problem. The survey closes on the 1st of June and the results will be published on the AMIS Yammer soon after. For any questions or ideas you can contact me at: carlijn.quik@amis.nl

*Required

Background

I will first ask you some questions about your background.

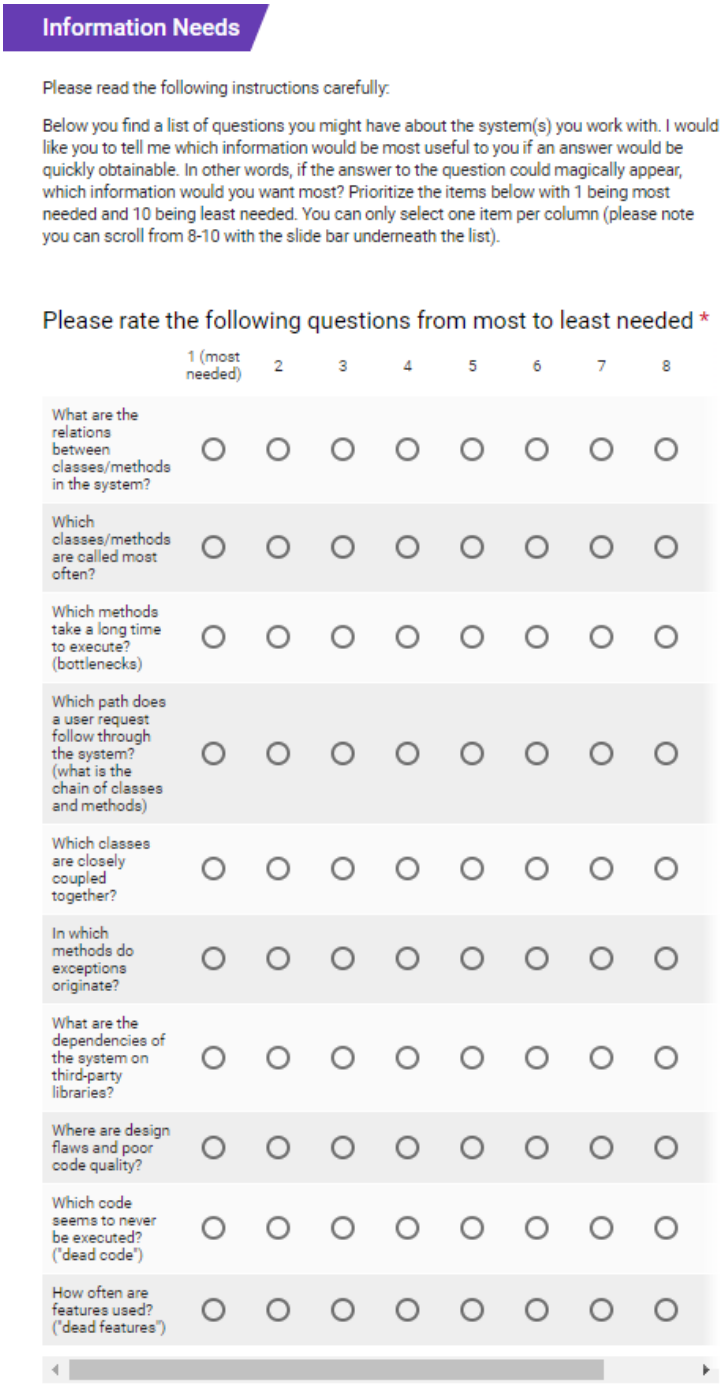
Which title best describes your role and tasks within the organisation? *

- DevOps Engineer
- Software Developer
- Software Architect
- Product Owner
- Administrator (System, Database, Platform or Applications)
- Other:

How long have you been working in a role similar to your current position (in years)? *

- 0-4
- 5-9
- 10-14
- 15-19
- 20+

Figure 55: *Survey part 2/3.*



Information Needs

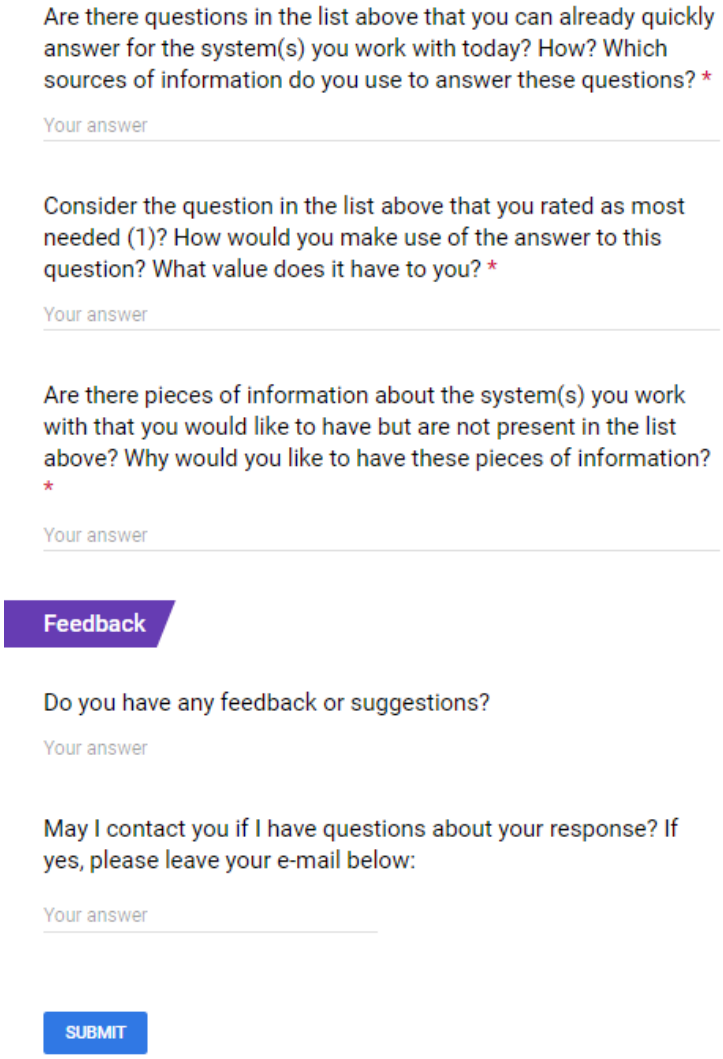
Please read the following instructions carefully:

Below you find a list of questions you might have about the system(s) you work with. I would like you to tell me which information would be most useful to you if an answer would be quickly obtainable. In other words, if the answer to the question could magically appear, which information would you want most? Prioritize the items below with 1 being most needed and 10 being least needed. You can only select one item per column (please note you can scroll from 8-10 with the slide bar underneath the list).

Please rate the following questions from most to least needed *

	1 (most needed)	2	3	4	5	6	7	8
What are the relations between classes/methods in the system?	○	○	○	○	○	○	○	○
Which classes/methods are called most often?	○	○	○	○	○	○	○	○
Which methods take a long time to execute? (bottlenecks)	○	○	○	○	○	○	○	○
Which path does a user request follow through the system? (what is the chain of classes and methods)	○	○	○	○	○	○	○	○
Which classes are closely coupled together?	○	○	○	○	○	○	○	○
In which methods do exceptions originate?	○	○	○	○	○	○	○	○
What are the dependencies of the system on third-party libraries?	○	○	○	○	○	○	○	○
Where are design flaws and poor code quality?	○	○	○	○	○	○	○	○
Which code seems to never be executed? ("dead code")	○	○	○	○	○	○	○	○
How often are features used? ("dead features")	○	○	○	○	○	○	○	○

Figure 56: *Survey part 3/3.*



A Appendix 4: Instrumenting Java applications with AjpoLog

AjpoLog stands for AspectJ partially ordered Logging. It was created by Tijmen de Jong (2019) [8].

Step 1: Download AjpoLog: The source code of the aspects can be found at [37]. Download it and save it in a folder called "trace-aspects".

Step 2: Add the plugin: In the pom.xml file that contains the plugin management of your application, add the following plugin:

```
<build>
        <pluginManagement>
        <plugins>
                <plugin>
                        <groupId>org.codehaus.mojo</groupId>
                        <artifactId>aspectj-maven-plugin</artifactId>
                        <version>1.11</version>
                <configuration>
                        <complianceLevel>8</complianceLevel>
                        <source>8</source>
                        <target>8</target>
                        <showWeaveInfo>false</showWeaveInfo>
                        <verbose>false</verbose>
                        <Xlint>ignore</Xlint>
                        <encoding>UTF-8</encoding>
                <!Optional>
                        <excludes>
                        <exclude>**/*.java</exclude>
                        </excludes>
                        <forceAjcCompile>true</forceAjcCompile>
                        <sources/>
                        <!Optional>
                <aspectLibraries>
                        <aspectLibrary>
                        <groupId>your.groupID</groupId>
                        <artifactId>trace-aspects</artifactId>
                        </aspectLibrary>
                        </aspectLibraries>
                </configuration>
        <executions>
                        <execution>
                                <id>default-compile</id>
                                <phase>process-classes</phase>
                                <goals>
                                <! use this goal to weave all your main classes >
                                        <goal>compile</goal>
                                </goals>
                                <configuration>
                                        <weaveDirectories>
                                                <weaveDirectory>${project.build.directory}classes </weaveDi-
rectory>                                            </weaveDirectories>
                                </configuration>
                        </execution>
                        <execution>
                                <id>default-testCompile</id>
                                <phase>process-test-classes</phase>
```

```
                        <goals>
                                <!– use this goal to weave all your test classes –>
                                <goal>test-compile</goal>
                        </goals>
                        <configuration>
                                <weaveDirectories>
                                        <weaveDirectory>${project.build.directory}test-classes
</weaveDirectory>                                       </weaveDirectories>
                        </configuration>
                </execution>
        </executions>
        </plugin>
    </plugins>
    </pluginManagement>
</build>
```

Step 3: Add the references: In the pom.xml files of all sub-applications, add a reference to the plugin:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>aspectj-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Step 4: Add the logger: In the Log4j.xml file of your application, add the appender and logger. Make sure the file the logger writes to is a file that you can access.

```
<appender name="AJPOLog-FILE" class="org.apache.log4j.DailyRollingFileAppender">
    <param name="append" value="true" />
    <param name="encoding" value="UTF-8" />
    <param name="File" value="${catalina.base}/logs/AJPOLog-&appName;.log"/>
    <param name="DatePattern" value="'.'yyyy-MM-dd" />
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="="%-25d{ISO8601};[%t];%m
n" />
    </layout>
</appender>

<logger name="AJPOLog" additivity="false">
    <level value="DEBUG"/>
    <appender-ref ref="AJPOLog-FILE" />
</logger>
```

A JSON Structure

Figure 57: *Structure of the JSON Object used by the dashboard.*

```
{"nodes": [
    {"name": "org.architecturemining.program.example.band.Band",
    "parent": "org.architecturemining.program.example.band",
    "root": "org",
    "count": 1},

    {"name": "org.architecturemining.program.example.band.Song",
    "parent": "org.architecturemining.program.example.band",
    "root": "org",
    "count": 12},

    {node3},{node4},etc.],

"links": [
    {"source": "org.architecturemining.program.example.band.Band",
     "target": "org.architecturemining.program.example.band.Song",
     "count": 3,
     "sum_calls": 0.001, (sum of the duration of calls over this
        link)
     "calls":
         [{"startDate": "2019-08-08",
         "startTime": "12:14:30.273000",
         "endDate": "2019-08-08",
         "endTime": "12:14:30.273000",
         "callerID": "CallerPsuedoId: 1920387277",
         "calleeID": "CallerPsuedoId: 775931202",
         "source":
            "org.architecturemining.program.example.band.Band",
         "target":
            "org.architecturemining.program.example.band.Song",
         "thread": "[main]",
         "message": "public java.lang.String
            org.architecturemining.program.example.band.Song.getName()",
            (method called)
         "duration": 0.0,
         "duration_sum": 0.001, (sum of the duration of calls of
            this type)
         "count": 3, (number of times calls of this type are made)
         "avg_duration": 0.0003333333333333333, (duration_sum
            divided by count)
         "sub_calls": [
            {sub-call1},{sub-call2},etc.]},
        {call2},{call3},etc.]
    },{link2},{link3},etc.]
 }
```

A Appendix 5: Running the visualisation

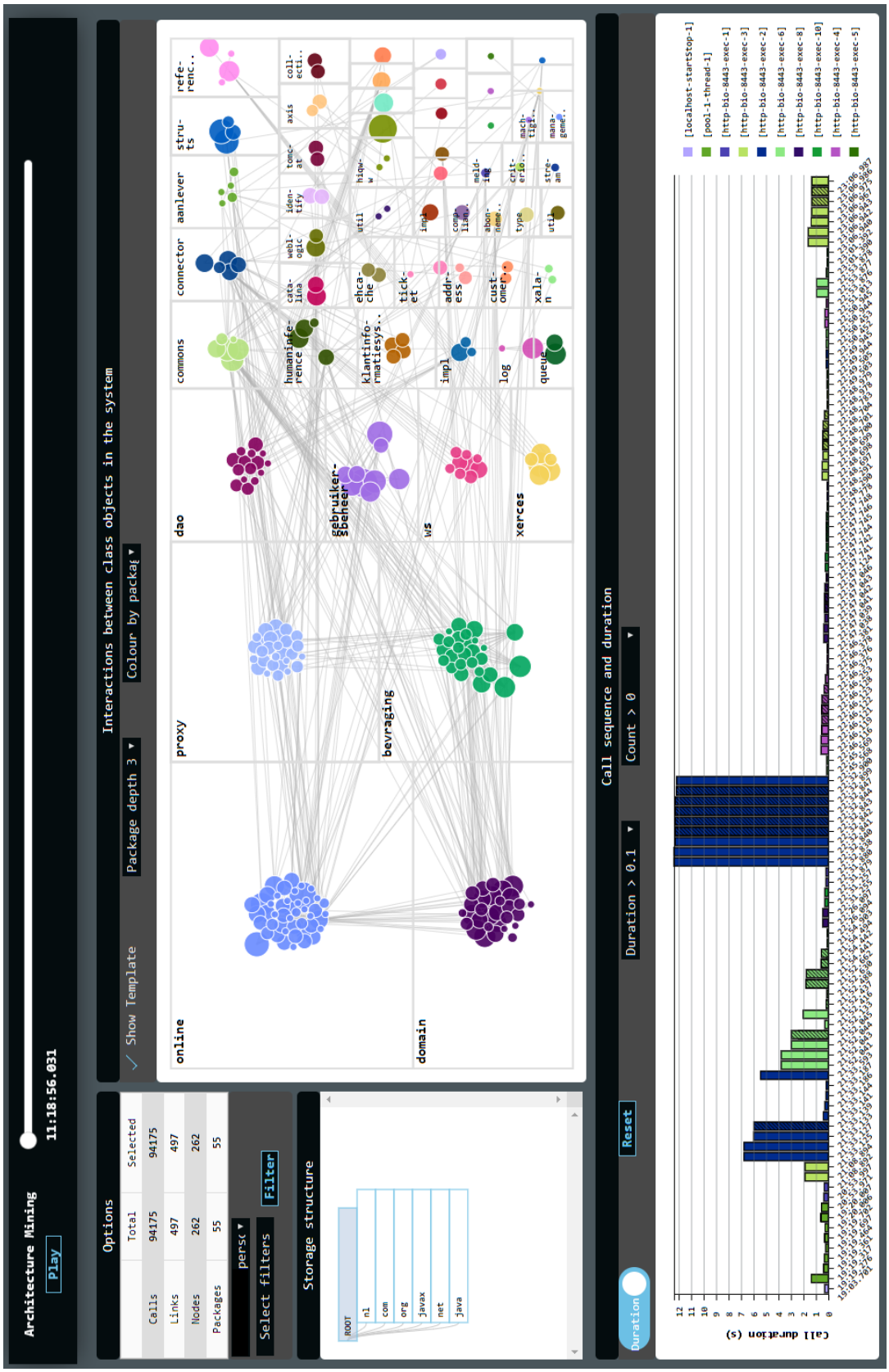
After producing the log files using AjpoLog, the following steps can be taken to run the visualisation:

1. Download the source code by executing "git clone https://github.com/carlijnq/architecture-visualisation.git" in the command prompt.
2. In the downloaded folder, there is a sub-folder called "datasets". Place the log files in this folder.
3. To convert the log files to the right json format that the visualisation uses, run "python data-processing/main.py datasets/filename.log", inputting the file that you would like to convert.
4. If the filename was not present in the visualisation before, add it to the file selection drop-down on line 152 in the index.html. For example: <option value="band">BandExample scenario 1</option>, make sure that "value=filename" corresponds to the name of the JSON file.
5. Now run the visualisation by executing "http-server" in the command line that is set to the location of the root folder of the source code ("architecture-visualisation"), and go to the IP address of the started server.

It is important to note that the log file has to be formatted in the following way: “date time ;[thread];type;caller;callee;message”. An example: "2019-08-08 12:14:30,248 ;[main];Entry;org.architectureremining.program.example.band.BandPractice.Static; java.util.ArrayList.CallerPsuedoId: 2092769598;public boolean java.util.ArrayList.add(java.lang.Object)". The format in which the logger writes to the log file is specified in the "log4j.xml" file as described in Appendix A, for example: "<param name="ConversionPattern" value="="%-25d{ISO8601};[%t];%m
n" />". This can be edited, but the files "log_to_csv.py" and "csv_to_json.py" then have to be changed too to be able to run correctly with that type of formatting.

A Appendix 6: Dashboard Design Enlarged

Figure 58: *Overview of the dashboard design applied to a scenario with 94,175 calls.*



A Appendix 7: Validation Protocol: Interviews

Thank you for making the time for this interview. The goal of this interview is to test whether the visualisation that I will show you aids you in answering a set of questions. For result processing purposes, I will record this session. Do you have any questions before we start?

Part 1: Questions for the not previously interviewed participants
To begin with, I would like to ask you some questions about your role and experience.

1. How long have you been working with the system(s)?
2. Do you have a title for your position? What is it?
3. How long have you been working in a position like this one?
4. Do you have any experience with software design (CASE tools? UML?) If yes, how long have you been working with software design?

Part 2: Questions without the dashboard
The participant is asked how he or she would answer the following questions with the tools that are currently available to him/her:

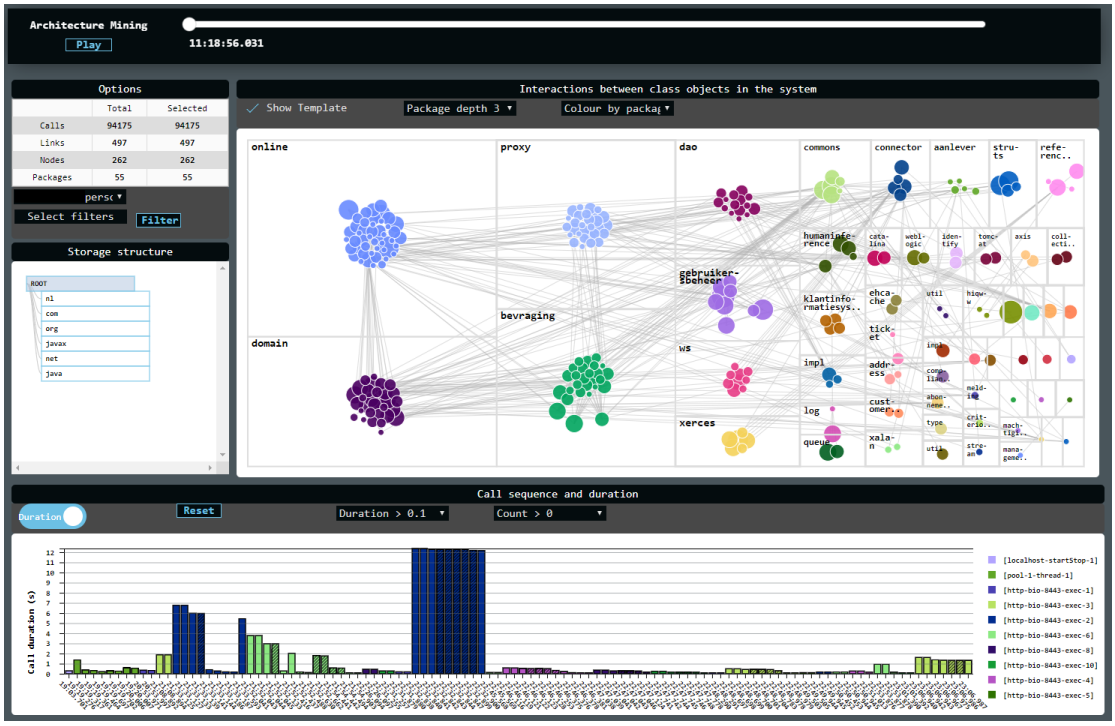
1. Which part(s) of the system contain calls that take relatively longer than the other calls to execute?
2. The calls that take a long time: how often are those called?
3. The calls that take a long time: which path do those follow through the system?
4. Which classes are called most often (and therefore important)?
5. What are the run-time dependencies of the system on (third-party) libraries?

Part 3: Questions with the dashboard A short demonstration of the dashboard is given. Then part 2 is repeated for a specific scenario, while the participant can make use of the dashboard. The chosen scenario is shown and explained to the participant using screenshots of the system. The scenario has the following steps:

1. Open form
2. Search for a company
3. Select the company of interest
4. Click search without selecting any filters
5. Confirm search to be done by clicking search again
6. Open one of the tabs with details about the company

The resulting log file contains 94,175 calls. Out of the 348 packages in the system, 55 were used within the scenario. Besides that, 262 out of 1,921 classes were used. The 94,175 calls made resulted in 497 unique (bi-)directional links between class objects. The dashboard using this scenario as an input is shown in Figure 59.

Figure 59: *Overview of the dashboard design applied to the selected scenario.*



1. Which part(s) of the system contain calls that take relatively longer than the other calls to execute? If there are multiple name three.
2. The calls that take a long time: how often are those called? If there are multiple calls name three.
3. The calls that take a long time: which path do those follow through the system? If there are multiple calls provide an answer for one of them
4. Which classes are called most often (and therefore important)? If there are multiple classes name three.
5. What are the run-time dependencies of the system on (third-party) libraries? If there are multiple libraries name three.

Part 4: Perceived usefulness

1. Do you think the dashboard provided a complete answer to the questions?
2. Do you think the dashboard provides you with insights you would otherwise not have?
3. Do you think using the dashboard would enable you to answer the questions more quickly?
4. Do you think you would use the dashboard in your daily work? What would you use it for? Do you think the dashboard would make your job easier?

Part 5: Perceived ease of use

1. Do you think it is easy to learn how to use the dashboard?
2. Once you know how to use it, do you think the dashboard is easy to use? Is it clear and understandable?
3. Which information took you the most effort to retrieve?
4. Did the dashboard behave in an unexpected way while using it?

Part 6: Overall rating

1. What do you think are the three most positive aspects of the dashboard?
2. What do you think are the three most negative aspects of the dashboard?

A Appendix 8: Validation Protocol: Focus Group

Thank you for making the time for this focus group. The goal of this focus group is to discuss the results of the visualisation(s). For result processing purposes, I will record this session. Do you have any questions before we start?

Part 1: General questions
I would first like to discuss some general patterns I have found within the visualisations.

1. I have noticed that a certain proxy number always represents the same class being called, do you have an explanation for this? For example, $Proxy48 always represents the class Bevragingservice.

2. I have noticed that the callee class of a call does not always conform to the class in which the called method is being called, do you have an explanation for this? For example, the call to the method onExecute() has as a callee the class SearchBedrijvenAction(), but the method is called in the class BaseAction().

Part 2: Discussing three scenarios
I would now like to ask you some questions about results that I have found within the scenarios. I will first ask the DevOps Engineers to reply, after which I will ask the Architect to reply. The reason for this is that I want to avoid groupthink ('a mode of thinking that people engage in when they are deeply involved in a cohesive group, when members striving for unanimity override their motivation to realistically appraised alternative courses of action' [92]).

Scenario 1: Scenario used during the interviews (app. 1)

1. In the scenario we discussed during the interviews, I noticed that the other application "application 2" is used, do you have an explanation for this?

2. Besides this, I noticed that the interactions between the class "SelfPopulatingCache" and "List-CacheService" take up relatively much time, do you have an explanation for this?

3. Are there other results that you find noticeable?

Scenario 2: Request details (app. 2)
The second scenario selected has the following steps:

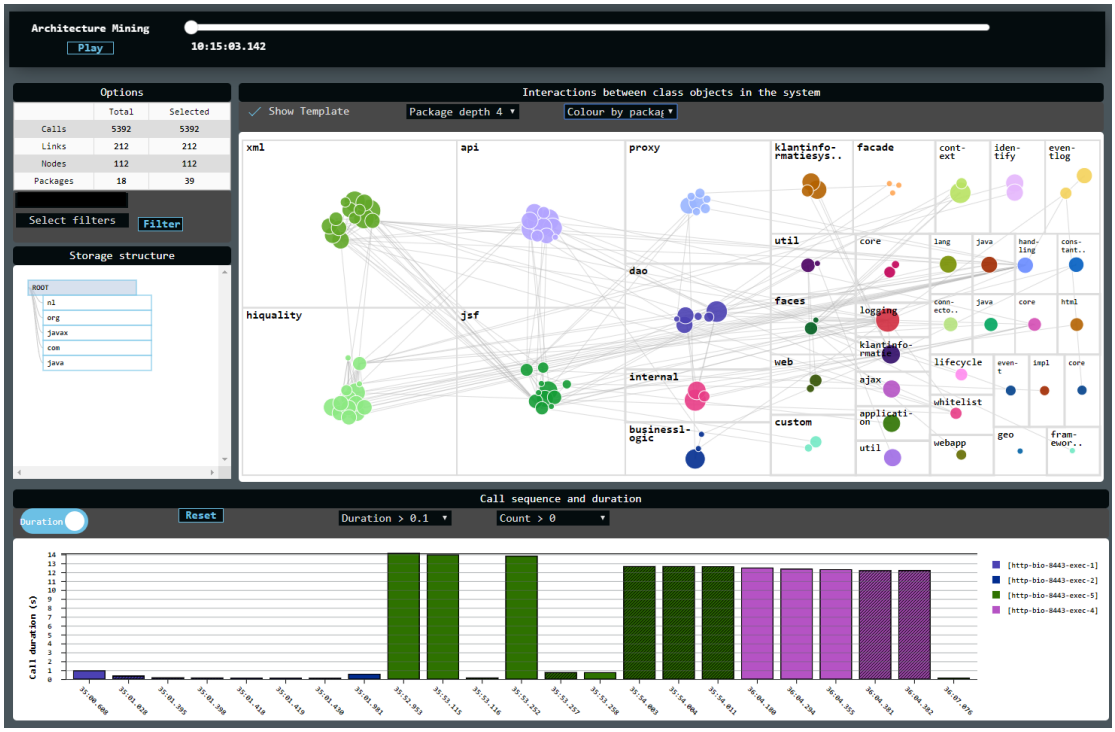
1. Open form

2. Fill out request form

3. Click on the request button

The resulting log file contains 5,392 calls. Out of the 91 packages in the system, 18 were used within the scenario. Besides that, 112 out of 1921 classes were used. The 5,392 calls made resulted in 212 unique (bi-)directional links between class objects. The resulting dashboard is shown in Figure 60.

Figure 60: *Overview of the dashboard design applied to scenario 2, package depth 4.*



1. In the scenario "request details", I noticed that the method getListFieldLabel() is being called 152 times, do you have an explanation for this?
2. I also noticed that "hiquality" is the most used package, do you have an explanation for this?
3. Are there other results that you find noticeable?

Scenario 3: Upload check (app. 1)
The third scenario selected has the following steps:

1. Click the upload button
2. Select the file to upload from your computer

This scenario is chosen because the DevOps team has noticed that executing this scenario is slow, and they do not know why it is slow. The resulting log file contains 37,486. Out of the 348 packages in the system, 37 were used within the scenario. Besides that, 98 out of 1,921 classes were used. The 37,486 calls made resulted in 129 unique (bi-)directional links between class objects. The resulting dashboard is shown in Figure 61.

Figure 61: *Overview of the dashboard design applied to scenario 3, package depth 3.*

1. In the scenario "upload check", I noticed that the method listBedrijven(List) takes a long time (9.025s - 3.902s). Do you have an explanation for this?

2. I also noticed that the class "menucomponent" is used a lot (1368 times), do you have an explanation for this?

3. Are there other results that you find noticeable?