Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Visualization of Performance Antipattern Detection Results

Matthias Popp

Course of Study:	Softwaretechnik
Examiner:	Dr.-Ing. André van Hoorn
Supervisor:	Dr. Dušan Okanović, Teerat Pitakrat, M.Sc., Jun.-Prof. Dr. Fabian Beck (Universität Duisburg-Essen)
Commenced:	July 3, 2017
Completed:	January 3, 2018
CR-Classification:	D.4.8, H.5.2

Abstract

In many cases developers do not understand appearing performance problems after the implementation or some changes. The problems can often be traced back to same root causes, which are present in many different software systems. Some of these problems are collected and analyzed, called anti-patterns, and can be detected by software diagnosis tools. Thus, organizations are using application performance management (APM) tools, to detect bottlenecks and other performance problems in their product. To support the developers, this thesis will deal with visualization of detected anti-patterns. Information about the anti-pattern, individual data from the system under test (SUT) to understand the problem and their solutions are a part of the report. The result of the work will be a prototype framework, which uses available diagnosis and APM results to generate the visualized based report, e.g. stack traces and time series.

Kurzfassung

Oftmals verstehen Entwickler nicht, wieso ihre Softwaresysteme Performance-Probleme aufweisen, nachdem sie es implementiert oder verändert haben. Diese Probleme können meistens auf schon bekannte Ursachen zurückgeführt werden. Die bekannten Ursachen, auch als Anti-Pattern bekannt, können von Software-Diagnosewerkzeuge erkannt werden. Solche Programme nutzen Messdaten von Application Performance Management (APM) Systemen und werten diese aus. Sobald ein Performance-Problem gefunden wird, werden die Ergebnisse in Form eines Berichts präsentiert. In dieser Arbeit geht es um die Erweiterung dieser Informationsberichte. Um die Entwickler daringehend zu unterstützen, dass sie das Performance-Problem verstehen und die Ursache herausfinden können, wird in dieser Arbeit versucht, solche Daten visuell darzustellen. Das Ergebnis dieser Arbeit wird ein Prototyp sein, welcher visuelle Berichte für gefundene Anti-Pattern erstellt.

Contents

List of Figures

List of Tables

List of Acronyms

APM application performance management

SPAs Software Performance Anti-Patterns

SUT system under test

List of Listings

Chapter 1

Introduction

This thesis investigates approaches to visualize performance anti-pattern results.

This chapter clarifies the motivation behind this approach and shows the goals of the thesis. The first Section 1.1 describes the motivation in detail. Section 1.2 gives an overview about the thesis goal and Section 1.2 describes the thesis structure and provides a short outlook.

1.1. Motivation

Nowadays, software systems are monitored by application performance management (APM) to prevent and detect performance problems earlier [HHMO17]. Performance problems, like little increases in load times, can have huge impacts of the profitability on a company. Therefore, companies should always pay attention to the performance of their systems.

The basic functions of performance supervision tools are often, visualization of performance-relevant measures, alerting or in some cases detection of problem instances. This performance problem instances are caused by design mistakes. Such issues have been documented in the form of performance anti-pattern.

These anti-patterns can be detected by APM tools. The common practices is to alert the user after an anti-pattern is detected. A report about the performance problem is then generated. The report consist the problem instance and a natural language description. However, users can't comprehend the detection or even the root causes. Trusting the results or looking in the measurements and analyzing these manually are the only possibilities to confirm the result. In order to improve this situation, this thesis

will use existing visualization approaches to add more visualized data to the detection report, depending on the detected performance anti-pattern.

Not every visualization technique is equally good. Each of them has its own qualities and can sometimes only be used to a specific scenario. After a short overview about existing visualization techniques in Section 2.4 and a more detailed description for the used techniques in Chapter 3, we will evaluate the techniques based on the data types and the presented anti-pattern. Therefore, the visualization techniques also have an evaluation in the performance visualization scenario.

1.2. Goal

The goal of this thesis is to implement a prototype reporting framework, which can be integrated into existing APM tools. It obtains information about the diagnosed anti-pattern and data from the APM tool to inform the user, with different visualization types, about the anti-patterns. The specific goals are listed in the following.

Visualization techniques research

Heger et al. [HHMO17] already mention that APM information needs to be presented in a meaningful and comprehensible way and can be categorized in business and technical scopes with different levels of abstraction. We will focus on the technical scope and a lower level of abstraction, because only on lower levels anti-pattern can be detected. A presentation of the server health, a higher abstract level, is useless to understand anti-patterns. Moreover, data types must be analyzed to exclude unrealizable visualization techniques on the one hand and to use existing techniques categorized by data types on the other hand.

The visualization techniques will also be analyzed and categorized. The categorization is based on the types of goals, which the performance visualization techniques can address [IGJG14].

Anti-pattern Visualization

Before we can start implementing the prototype we have to find a mapping between the visualization techniques, which we have collect before, and the anti-pattern. The goal is to find visualization techniques, which visualize the behavior of the anti-pattern and support the root cause detection. Therefore, two aspect are necessary. The first aspect is

the performance problem, which causes the anti-patterns. The second is the root cause, which caused the anti-pattern. Both aspect should be presented in one report.

Developing prototype

To generate a visualization report out of a detected anti-pattern, we have to develop a prototype. Existing implementations are used to support large amount of visualization techniques, which can adjust to the circumstances. Therefore, the prototype should use web frameworks, because the web is the most common way to present data and there are the most existing visualization libraries. We receive the measurement data from the APM. This data will be saved as execution traces or time series database systems. To receive data from database systems and other data formats, an interface is required.

Evaluation

After the implementation, the result is then evaluated in a user study. The selected visualization techniques and the benefits should be rated for each individual anti-pattern. Therefore, the participants should have fundamentals of anti-patterns or at least a description of the anti-patterns.

Thesis Structure

This thesis is structured as follows:

Chapter 2 – Foundations and State of the Art: In this chapter, the foundations for this thesis are described. This includes related work and the state of art according to this thesis. For a better understanding of the anti-pattern detection, an introduction to the Software Performance Anti-Patterns (SPAs) and APM are given. A more detailed description is given for the visualization foundations and the different data types.

Chapter 3 – Anti-pattern Visualization: The combining of visualization and anti- pattern are presented in this chapter. In the following, suitable visualization approaches are investigated for each analyzed anti-pattern.

Chapter 4 – Implemented Visualization Tool: The implementation of the framework prototype and the integration in the existing environment are outlined in this Chapter. The used tools and technology are also presented.

Chapter 5 – Evaluation: To evaluate the prototype, a user study is used, which is described in this chapter. Participants will be presented the prototype and be asked questions.

Chapter 6 – Conclusion: In the last chapter, we will sum up the outcomes of this thesis and outline the future work.

Chapter 2

Foundations and State of the Art

This chapter introduces the used technologies and the environment, which are related to the work of the thesis. Section 2.1 provides how APM tools work and in Section 2.2 what are performance anti-patterns. To better understand important terms, in the state of the art part an overview about visualization techniques (Section 2.4) and data types (Section 2.3) is given. In the next part, Section 2.5, related work to this thesis is provided.

2.1. Application Performance Management

An APM tool "aims to achieve an adequate level of performance during operations" [HHMO17]. It covers two main functions. The first one is to continuously monitor the current state of the software system. The second is to detect, diagnose and resolve performance related problems [HHMO17]. It collects data from different system levels, e.g. application or hardware, and uses therefore different techniques, e.g. injected code or log analysis. Two data types are commonly used: time series (Section 2.1.2) and execution traces (Section 2.1.3). These data can be used to extract architectural information, including logical and physical deployments. APM uses different views to represent the information in a meaningful way. The storage data can be interpreted and used to solves following goals: *problem detection and alerting*, *problem diagnosis and root cause isolation (Section 2.1.1)* and *system refactoring and adaptation*.

2.1.1. diagnoseIT

diagnoseIT [HHO+16] is an APM tool and focused on problem diagnosis and root cause isolation. It receives execution traces and uses expert knowledge for analysis. Due

to the use of OPEN.xtrace, detailed in Section 2.1.3, diagnoseIT is independent from specific monitoring tools. After diagnoseIT receives traces, it uses expert created rules to analyze each trace. Based on the insights diagnoseIT obtains by applying rules, it detects potential anti-patterns. The result of this process is the problem instance, which holds the performance problem, and proposes solutions in natural language.

2.1.2. Time Series

Time series data is a two dimensioned data type and consist of a series of data points. Each data point describes one measurement and contains a timestamp, as an independent variable, and a dependent value, e.g. response time. Due to the constant continual time variable, time series are used in any domain of applied science and engineering, which involve temporal measurements. Often, time series databases are used to store time series and are optimized for this type of data.

2.1.3. Execution Traces

Execution traces capture information on a software system's. They contain runtime behavior, data on system-internal software control flows, performance, as well as request parameters and values [OHH+16]. Moreover, these are used for dynamic program analysis and, in its simplest form, the representation of the control flow of a method execution. Simple execution traces can be stored in a hierarchical data structure, except traces with loops. Therefore, common visualization types are tree-based types, e.g. dendrogram. Open Execution Trace Exchange (OPEN.XTRACE) is a data format for simplification of data interoperability and exchange between APM tools [OHH+16]. Furthermore, it serves as an abstraction layer between diagnoseIT and the APM tool.

2.2. Software Performance Anti-Patterns

In this Section, we describe Software Performance Anti-Patterns (SPAs). SPAs are documented design mistakes, that are consistently made by developers. These are similar to software design patterns, which document in contrast best practices [Par07]. Because some SPAs have similar symptoms and causes, Wert [Wer13] derived a taxonomy. Through the similar symptoms, the visualization of these SPAs are almost identical and the SPAs will be merge in this chapter, shown in Figure 2.1, based on Wert's taxonomy.

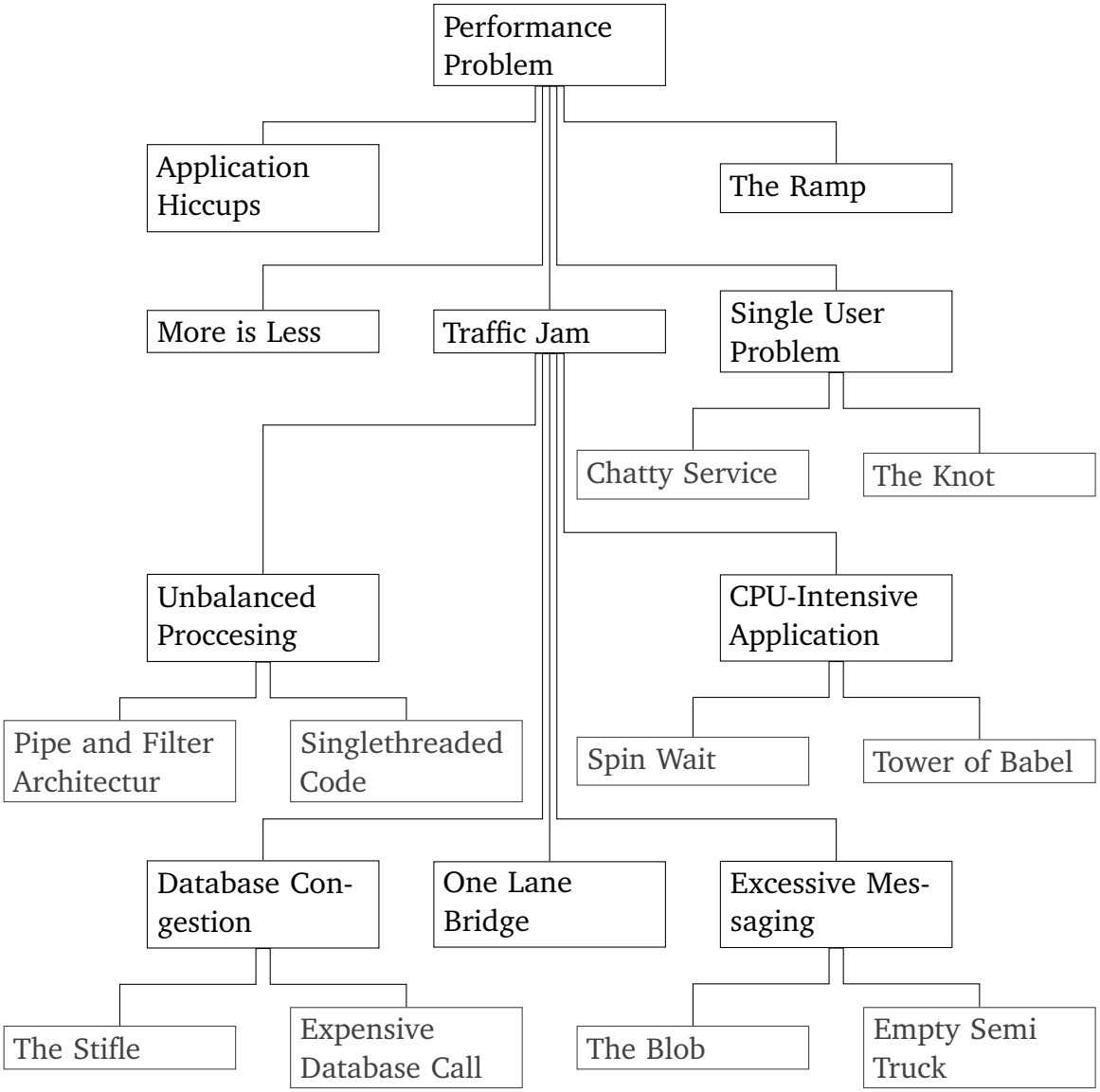


Figure 2.1.: Anti-pattern Taxonomy

Application Hiccups

The Application Hiccups anti-pattern describes recurring phases with high response time. To detect the anti-pattern, the observation time must be long enough to spot more than one hiccup. Hiccups caused by garbage collection are a typical cause for the Application Hiccups anti-pattern [Wer13].

The Ramp

The behavior of the Ramp anti-pattern is that the response time increases with the operation time of the system under test (SUT). Causes of the Ramp are often data storages,e.g. database or caches, which grow with the uptime of the SUT [Smi02].

Single User Problem

The Single User Problem comprises all types of performance problems that exhibit high response times under single-user load. The number of users does not influence the performance. The Knot and Chatty Service are two anti-pattern specifications. Potential root causes are, big amounts of service calls (Chatty Service) or expensive external service calls to perform a simple task (the Knot) [Wer13].

More is Less

The More is Less anti-pattern occurs, when too much processes run on the system and the system resources are exhausted. The operation system is busy with communicating with the hard disk, rather than processing requests [Smi02].

Traffic Jam

The Traffic Jam anti-pattern describes performance problems with wide variability response times caused by overloaded situations. The following anti-patterns can also be the causes for the Traffic Jam [SW02].

Unbalanced Processing

In case of The Unbalanced Processing anti-pattern, the workload of available CPUs is, in contrast to the performance, low. Typical root cause of the performance problem can be single-threaded code and an unevenly distribute Pipe and Filter Architecture with one filter becoming the bottleneck [Wer13].

CPU-intensive Application

The CPU-intensive Application anti-pattern groups all types of performance problems that lead to a high CPU utilization on the SUT [Wer13].Tower of Babel is one of this and its cause are too many data transformations into an exchange format [SW03]. Another one is Spin Wait and this anti-pattern describes the misuse of an empty loop for thread synchronization.

Database Congestion

The Stifle is one of the anti-patterns Database Congestion summed up. Database Congestion describes performance problems which are created by the communication and usage of the database system. For example, the cause of the Stifle is the sending of too many fine-grained database statements [Wer13].

One Lane Bridge

The One Lane Bridge anti-pattern is caused by a bottleneck service (e.g. database access, not multi-threaded processes, etc.) [SW00]. The symptoms are like the Traffic Jam anti-pattern.

Excessive Messaging

Excessive Messaging summed up the Blob and the Empty Semi Truck anti-pattern and describes inefficient communication between components and system nodes [Wer13]. If the SUT contains one single complex controller class that does most of the work, then the anti-pattern is called the Blob or the God Class [SW00]. The Empty Semi Truck occurs in systems where an excessive number of small messages are transmitted. The amount of additional overhead and preparation time outweigh a real task.

2.3. Data Types

The basic of each visualization is the data that is to be displayed. Because data comes from many and different sources and for various use cases we classify data types in this section. Data can be raw or it can be filtered, scaled or interpolated. Each data set can be defined as a list of n records, $(r_1, r_2, ..., r_n)$. Record r_i consists of m independent

(m_i) or dependent (m_d) variables, $(iv_1, iv_2, ...iv_{m_i}, dv_1, dv_2, ...dv_{m_d})$. A variable can be a number, a string or a complex structure. An independent variable iv_i consists a value which is not controlled or affected by another variable, e.g. time. Contrary to that, the value of a dependent variable dv_i, e.g., response time, is affected by other variables such as time [WGK10]. Furthermore, data set can be categorized in the following seven types [Kei02] and Shneiderman laid the foundations for this [Shn96].

2.3.1. One-Dimensional

In one-dimensional data sets each record has only one variable ($m = 1$). A typical record example is an array.

2.3.2. Two-Dimensional

Most of the time-series data are two-dimensional data. They have two variables ($m = 2$) and they can be dependent, e.g., time and response time, or independent, e.g., longitude and latitude.

2.3.3. Multidimensional

If a record has more than two variables ($m \geq 3$), it is a multidimensional data record. Each system with a database has one or mostly more multidimensional data sets. Every column in an entry is one dependent or independent variable.

2.3.4. Hierarchies data

The hierarchies data structure is equal to the multidimensional except for the links between two records. The tree data structure is a good example, each record has one parent record, except for the root record, and none or more children records. It follows, that the root record is in the highest hierarchy level and its children are one level lower and so on.

2.3.5. Text and Hypertext

Because not all information can be stored in numbers or other data structures, it is necessary to save information as text or hypertext. In the age of the world wide web, hypertext is becoming more and more important. The difference between text and hypertext is, that hypertext is not linear and has references to other texts.

2.3.6. Graphs

Graphs have similarities to Hierarchies data structure. A graph has nodes as variables and edges for the connections. The data structure can be used for large networks or the interaction between components in a software system.

2.3.7. Algorithms and Software

Algorithms and software data are not only texts or hypertexts. Every algorithm and software is in an context and fulfilling a purpose.

2.4. Visualization Techniques

Due to the big amount of visualization techniques, it is important to categorize and find a taxonomy for the different types. Our approach is to focus on the thesis problem [WL90] and organize the techniques in context-based groups. The context refers to anti-pattern detection strategies. Other approaches are to categorize based on the data structures types [WGK10] or fields of research.

Software Visualization techniques cover visualization types related to source code. This includes static (architecture) and dynamic data, such as call graphs and program flows [IGJG14].

Visualization types in the hardware context visualize data from performance indicators, like CPU usage or network traffic.

Task visualization consists of closer examination at individual task circumstances. Task are performed by higher level processes, threads and jobs.

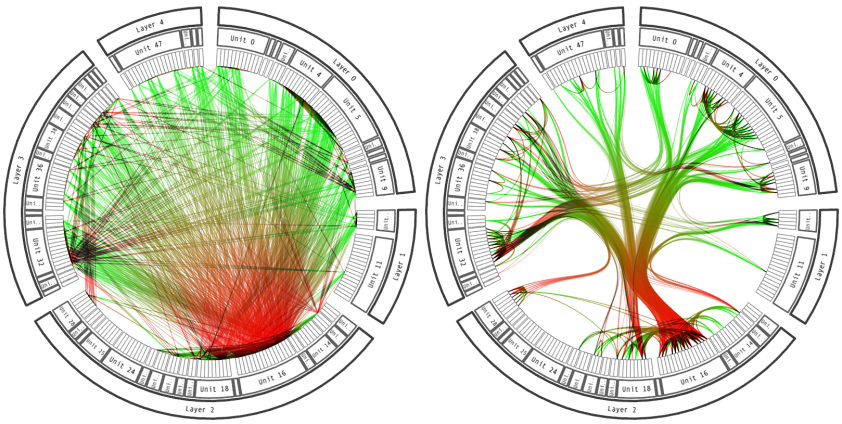


Figure 2.2.: *EXTRAVIS* Call relations within a program shown using linear edges (left) and using hierarchical edge bundles (right) [CHZ+07]

2.4.1. Software Visualization

Software visualization techniques are often used for debugging and education, such as *HeapViz* [AKG10] or Data Structure Visualizations [Gal]. Though these techniques are not applied for performance visualization, features can be adopted.

Serial Trace Visualization

Serial Trace Visualization shows a sequence of events. Particularly, execution traces from sizable programs are not easily understood, because the efficient visualization of both the structures and the many interrelationships is far away from trivial. There are some different approaches to visualize traces. The common practice is time based, one of the axis represents the time variable while the other shows different methods, processes or instructions. Pauw and Heisig et al. [PHD10] mapped system trace events, which contains time, event type, event subtype, event specific information, process id and system area, in the *Event Flow View* into two dimensions. Firstly, they mapped the time to the vertical dimension. This allows a user to see different phases of behavior over time (proceeding downward). Then they position each event based on the associated component on the horizontal dimension [PHD10].

Another approach is *EXTRAVIS* [CHZ+07]. As shown in Figure 2.2, the circular bundle view offers a detailed visualization of the system's structural entities and them interrelationships. The edges represent the method calls and the methods are placed in a circular layout.

lviz [WYH10] is another approach to visualize serial traces. This technique based on dot plots and each dot represent one or many events, if they are close to each other. It can be used for solving performance problems, program failure diagnosis and finding execution patterns/anomalies.

Call Graph Visualization

Call Graph Visualization is often used in debugging and helps to understand caller-callee relationships. Combined with other techniques call graphs can also be used for performance analysis.

The most common technique is to use nodes, which represent the function and the links between the nodes represent the function calls, known as node-link metaphor.

There are several tools that use an indented tree layout [ABF+10]. The colors of the nodes are often used to visualize duration time. Therefore, the horizontal space can be used to add tabular data or visualize the function.

The conventional tree visualization technique is also used. Due to the hierarchies data structure (Section 2.3.4) the nodes sorted from top, root call, to bottom. Some approaches, e.g. [LTOB10a], use also different abstraction levels to visualize large call graphs.

The Flame Graph is a new Approach to visualize a collection of stack traces [Gre16]. Each function in the stack trace is represented as a column of boxes. The y-axis shows the stack depth and the x-axis spans only the stack trace. The width of each box is relevant and shows the frequency at which that function was present in the stack traces.

Sunburts [AH10] and Treemaps [OJHS04] have also been used to represent call graphs. This visualization techniques are often implemented as interactive types and they use colored nodes to visualize the performance data.

Code and Code Structure Visualization

Code and Code Structure Visualization is often included in other approaches and shows the code on demand. For example, if a user clicks on a node in the call graph. The code is an important part to detect potential performance problems.

Seesoft [ES92] displays source code by representing files as rectangles whose height corresponds to the size of the file. Each code line is shown as row in the rectangles. They colored this rows in shade of red to visualize the *hot spots*.

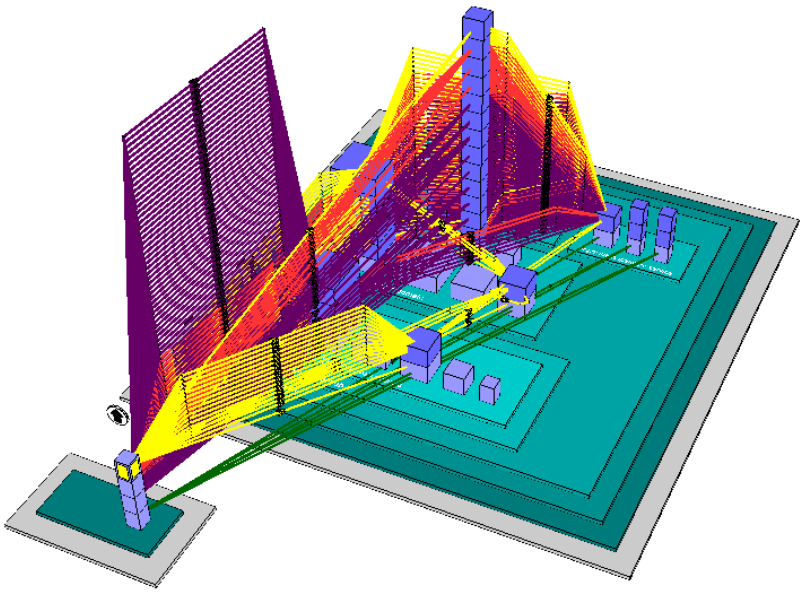


Figure 2.3.: *SynchroVis* [Döh12]

Another approach is to visualize software systems as cities. Wettel and Lanza describe the approach to visualize software with a 3D city metaphor [WL07]. *SynchroVis* [WWF+13] adopted the approach and added the visualization of the dynamic properties by monitoring the program traces. In Figure 2.3 *SynchroVis* is shown, the *districts* (dark cyan) represent the packages or components, the buildings are the classes (purple) and their instances (blue), and the links visualize the operation calls.

2.4.2. Hardware Visualization

To understand the behavior of the system, we should also visualize the hardware on which the code is running.

Large software systems are not running on one physical device, but on several nodes which are connected via network. Boxfish [LLB+12] uses 2- and 3-dimensional meshes for displaying network performance data, with nodes are vertices and links as edges. The 3D view is a simplified planar projection of the network traffic, so that edges do not overlap. This view is designed to allow a user to identify easily trends and patterns in the network traffic.

Processor-based techniques using typically histograms, bar or line charts to visualize performance data per processor. Line charts, bar charts and scatter plots are the most common data graphics [HBO10]. The simple line chart technique is based on the original

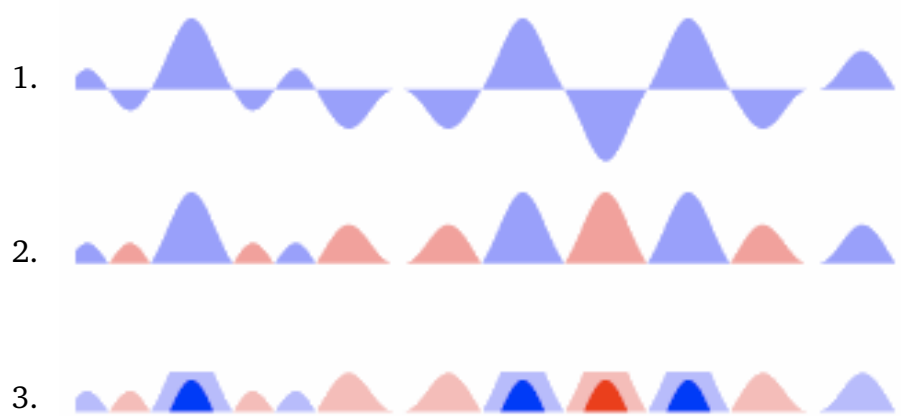


Figure 2.4.: Horizon Chart: 1) Filled line chart. 2) Flipped negative values over x-axis and colored red. 3) Divided into bands and overlaid, again halving the height [HKA09]

chart by Playfair [Pla86]. Time is mapped on the x axis horizontal and the value, of the two-dimension data, on thy x axis vertical. The mapped data points are connected by straight line segments. To smooth the line curve-fitting methods are used. Scatter plots are similar to line chart, but without a line.

An approach which break with conventional line charts Heer et al. [HKA09] create a space-efficient time-series visualization technique. The transformation from a filled line chart to a horizon chart is shown in Figure 2.4.

To represent the memory, the usage techniques can be adopted from the CPU usage techniques. There are several approaches that depict the allocations and deallocations over time [CFA+06] or depict the migration to the different cache levels [CR11]

2.4.3. Task Visualization

Execution traces and system logs are the main data source for task visualization. These data mostly contain timestamps, function calls, message receives and job initiations [IGJG14].

The majority of approaches assigns the time to the x- or y axis, represents the event of each task according to a row or column and cluster the events.

The De Pauw et al. visualization technique [DWB13] clusters the jobs by users and how the jobs consume the shared resources over time. They also mention, that their

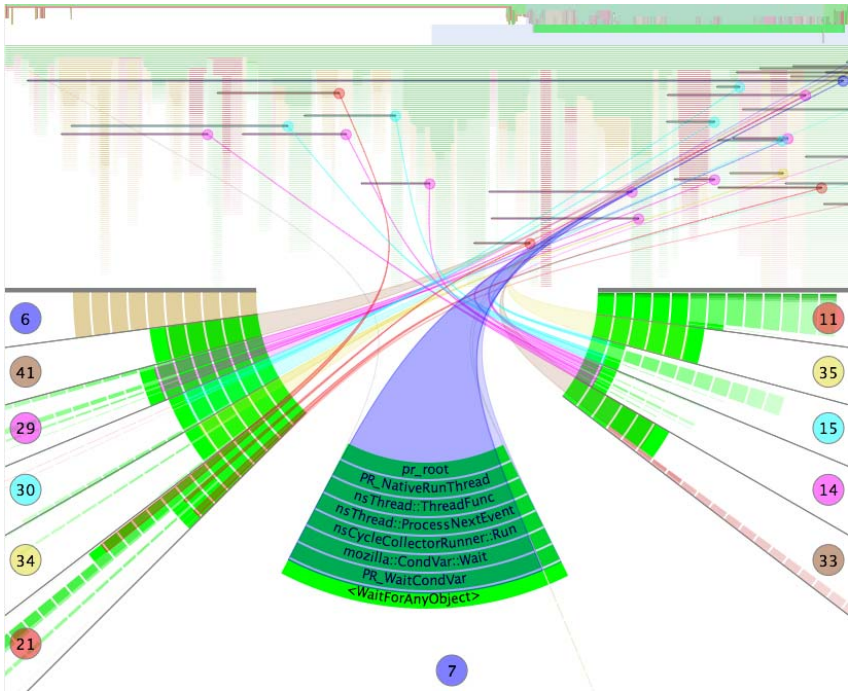


Figure 2.5.: *SyncTrace* [KTD13]

technique “shows trends in multiple variables over time much more clearly than standard time-based graphs” [DWB13].

SyncTrace create a visualization design to analysis threads runtime behavior and inter-relationship [KTD13], shown in Figure 2.5. The upper part gives an overview of the trace data, to identify the main synchronization relationships between two threads. At the bottom is a sunburst-like view which draws multiple threads as sectors of a circle and connect upper thread bars with the sectors.

Sigovan et al. [SMM13] present a visual analysis method for parallel communication traces. They are displaying concurrent function calls as particles moving on trajectories parallel to each other. With this approach they aim to make it easier to detect potential communication slowdowns, when particles that should be aligned fall out of synch [SMM13].

2.5. Related Work

Related work is every work which deals with data visualization. Some specific work which focuses on performance visualization is already mentioned in Section 2.4.

VAMPIR [WH96] is a visualization environment with similar functions as our prototype. It translates a given trace file into a variety of graphical views, e.g., state diagrams, activity charts, time-line displays, and statistics. Moreover, it can be used to locate performance bottlenecks and support interaction and filtering to reduce the amount displayed information.

PAVO [WKK16] is a framework for the visualization of performance analyses results. It provides the following features: *generic visualization for all stages of quantitative analysis*, *automated selection*, *slicing* and *diagram switching*. Moreover, PAVO supports line chart, bar chart, box plot, differences chart and pie chart. The main goal is that "the user specifies *what* he wants to know [...] and PAVO automates *how* two visualize it" [WKK16].

Most of the APM tools are also using visualization techniques to visualize there measurements or using monitoring platforms, like *Grafana* [Gra] or *Datadog* [Dat]. They using mostly time-series visualization techniques, e.g., line charts, stacked charts or heatmaps. Functions like overreaching zooming and details on demand are solid components. Furthermore, they support customize views, time-series database management system integration and open APIs.

Chapter 3

Anti-pattern Visualization

In this chapter, we would like to find visualization-types for all described Software Performance Anti-Patterns (SPAs) in Section 2.2.
First of all, we need to understand how diagnoseIT (Section 2.1.1) detected the SPAs and primarily which data the tool or a human needs to detect the root cause. Furthermore, the effects of the SPAs are also important. The users have to understand why the performance is impaired by the anti-pattern.
According to the most common visualization pipeline [PD10], shown in Figure 3.1, the data analysis on the raw data is done by the diagnoseIT rule engine. The filtering task is to collect the data from different sources and to prepare it for the mapping task. The preparation consist to cut the data to the significant part, e.g. the hiccup parts on the response time. In the mapping task, the data will be assigned to a visualization type. The last part of the pipeline is the rendering part, which mostly third-party frameworks, e.g., d3.js, MATLAB and Excel, does. From this follows that the main parts are filtering and mapping.

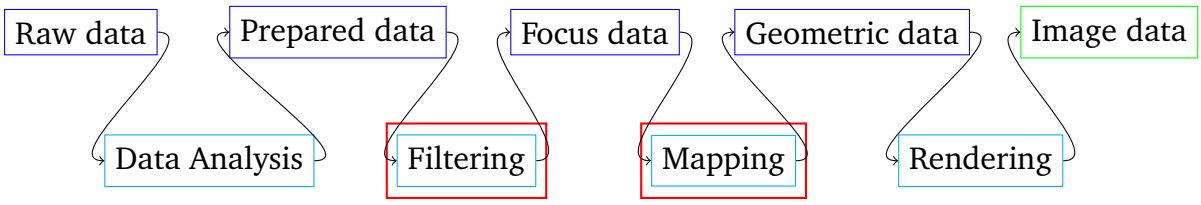


Figure 3.1.: Visualization Pipeline

Anti-pattern	**Data Types**	**Visualization Techniques**
Application Hiccups	time-series	line chart, horizon chart, spirals chart
The Ramp	time-series	line chart, stacked chart
Single User Problem	time-series, two-dimension	line chart, chrod (*EXTRAVIS*), dot plot (*lviz*)
More is Less	time-series	line chart
Traffic Jam	time-series, hierarchy	line-chart, icicle, sunburst
Unbalanced Processing	time-series, hierarchy	line-chart, stacked chart, call graph
CPU-intensive Application	time-series, text, multidimensional	matrix, line chart, text
Database Congestion	hierarchy, two dimensional	sankey, indented tree, flamegraph
One Lane Bridge	time-series, two-dimension	line chart, chrod, *SyncTrace*, flamegraph
Excessive Messaging	two-dimension	call graph, *SynchroVis*, graph

Table 3.1.: Anti-pattern overview

3.1. Application Hiccups

Temporarily system overloads by periodic tasks, e.g. garbage collector, increased the system response time. This behavior is called Application Hiccups anti-pattern and can be detect by monitoring the response time.

To detect the anti-pattern, only the response time is needed. There are two important variables. The first one is the highest response time, during the hiccup, which must be higher than a threshold. The second one is the duration of the hiccup, which does not exceed the specified proportion of the experiment duration. [Wer13].

We already mention that the response time is the most important identifier for the Application Hiccups SPAs. The response time is a two-dimensional (Section 2.3.2) data type with n records. Each record r_i has two variables, one independent and one dependent. The independent one is the time, given as timestamp. The dependent one is the response time, which depends on the time and is given as time unit.

Based on the data type we can adopt visualization techniques from processor and memory usage techniques in the hardware visualization section 2.4.2. Histograms or bar charts are not optimal for sequence data, e.g. timestamps, on the x-axis. Line charts

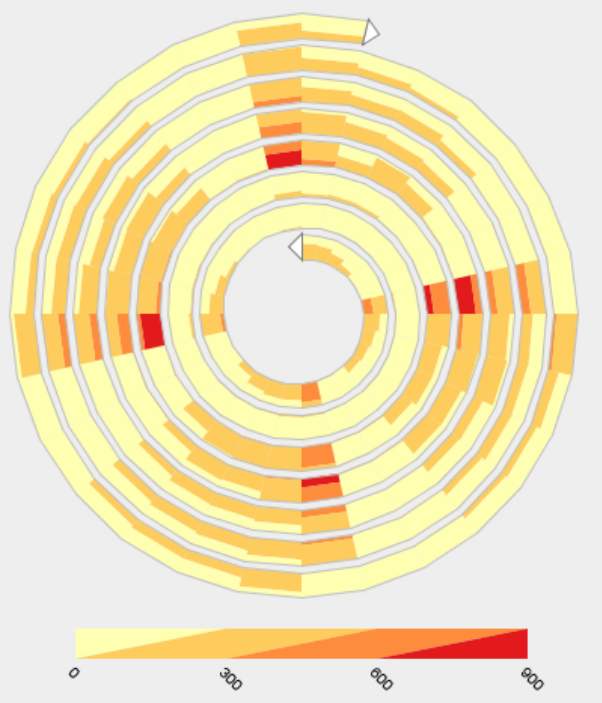


Figure 3.2.: Spiral Chart [TS08]

are in this case better, because they are used for detecting trends and patterns and, do not really intended to give people exact numbers.

Otherwise, we know that observation time can be long, so that the scaling is too high to visualize the anomalies significant. We can use the Horizon Chart and expanding the x-axis length by stacking the charts to reduce the scaling. Christian Tominski et al. [TS08] expand also the x-axis by using spirals. Shown in Figure 3.2, it is particularly useful for spotting seasonal patterns in time series data. The cycle length is configurable and can be set to time interval between two raises.

3.2. The Ramp

The Ramp anti-pattern occurs, when the processing time constant rises with the operation time. It may take several weeks until an increase reveals. Root causes are mostly growing databases or data structures.

Similar to the Application Hiccups anti-pattern the Ramp can be detect by monitoring the response time. A regression line can be laid through the data points and if the slope of the line is positive and higher than a threshold, the Ramp is detected [HN16].

Figure 3.3.: Stacked Chart visualization for 4-time series [JME10]

Due to the same detection data, the data type of the visualization data are similar to the Application Hiccups anti-pattern. However, the difference is that dot or line charts are optimal, because the rise is also visible independent of the scaling. If we use a dot chart a possibility would be to visualize the regression line to underline the rise. Through that the Sisyphus Database Retrieval anti-pattern can be the cause of the Ramp it can be important to visualize the database and the database interaction. The Sisyphus Database Retrieval anti-pattern describes the grows of the time for processing the database queries, because of the big amount of data in the database. Therefore, it would be exciting to also present the query processing time and the database size over time. This data can be visualized in separate line charts or combined in one chart.

Javed [JME10] et al. evaluate shared-space techniques and small multiples graphs based on time series data. Their results show that the small multiples graphs are efficient for data with a large visual span. On the other hand, shared-space techniques are more efficient for data where the impact of overlap is reduced.

Based on this and the possibility to scaling the x-axis we prefer to use a shared-space technique, like the standard line chart or stacked chart (Figure 3.3).

3.3. Single User Problem

The Single User Problem describes an anti-pattern which is not influenced by the amount of users. Both root causes the Knot and the Chatty Service are based on bad service calls. The consequences of this anti-patterns, like the most, are high response time, but even in cases of low load on the system under test (SUT).

These anti-patterns can be detected by monitoring external service calls. In Section 2.4.1 we describe techniques to visualize a sequence of events, which are in this context service calls. By the fact that time is of no importance for understanding the problem, we can exclude the time-based approaches. Furthermore, *lviz* [WYH10] the dot plots approach is a common used technique for representation of behavior of the persistent actions of an application, interactions between multiple applications, and the functioning of the system as a whole. The approach, by Cornelissen et al. *EXTRAVIS* [CHZ+07], to visualize the events as edges and receiver or sender on a circular bundle, can be used to represent the service calls. The view has two more variables which can be used for visualize more information. The first one is edge color, or rather color gradient, and can

be used for indicating the direction or the response time of each call. The second one is the thickness of a spline, which indicates the number of calls between two elements. Another function we can adopt is the zooming. In *EXTRAVIS* the user can select an interval and thus reduce the timeframe under consideration. Because the Knot and the Chatty Service anti-patterns perform under single and simple tasks, we can use the zooming to focus on individual tasks.

The second interesting data we can present is the independence of load and response time over time. Already described in Section 3.2 we would use a simple line chart visualization for two time-series, number of users and response time.

3.4. More is Less

The More is Less anti-pattern describes a performance decrease, when the SUT tries to accomplish more work than the resources allow. A cause can be, too many database connections, allowing too many request or creating to many pooled resources.

Response time and CPU time are required data points from the database to analyze the performance problem. The decisive factor here is the ratio. If the response time higher than the average, but the ratio is lower than the average ratio, we can assume an overload situation. By the fact that the CPU time can be only one cause, it is interesting to have a look at the memory and the running threads.

The data types of the memory and the running threads are also two dimensions and the independent variable is time. Therefore, we can adopt the already described line chart techniques by Waqas Javed et al. [JME10].

3.5. Traffic Jam

The Traffic Jam anti-pattern describes high variance of response time. On overload situations some request get stuck in "traffic" and others are not effected.

To detect the Traffic Jam, the coefficient of variation is used [HN16]. The coefficient of variation is often expressed as a percentage, and is defined as the ratio of the standard deviation to the mean. When the value exceeds the threshold, the Traffic Jam anti-pattern is detected.

Similar to the other anti-pattern response time techniques, we can use a line chart again. To focus on the variance, we can manipulate the x-axis and set it to the response time average or set different colors depending on the ratio to the average. This underlines the

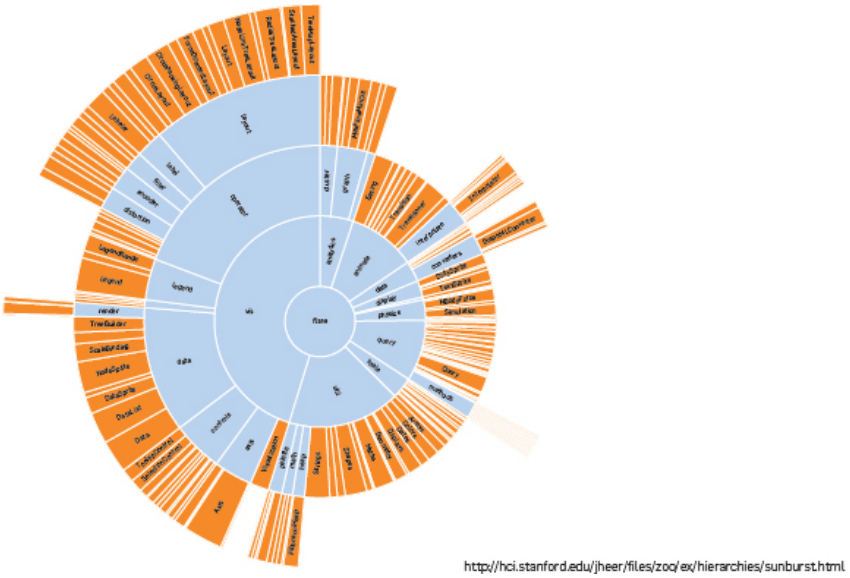


Figure 3.4.: Sunburst [HBO10]

high response time fluctuations. Based on the anti-pattern description, different request have different response time, we can use a filter to visualize response time course per tasks. The user can interact and limit the causes.

Parallel to the singe request we visualize the depending execution trace. Common visualization types for hierarchy data (Section 2.3.4) are adjacency diagrams [HBO10], like sunbursts or icicle tree layouts. Nodes are drawn as solid areas and their placement is relative to the adjacent nodes, shown in Figure 3.4. Moreover, in contrast to other hierarchies data visualization techniques, we can encode the response time to the length of areas and get an additional dimension.

3.6. Unbalanced Processing

The Unbalanced Processing occurs when the work is not evenly distributed among available processors [Wer13]. Single-threaded Code and Pipe and Filter Architecture are two root causes of the Unbalanced Processing symptom.

By monitoring the utilization of the CPUs and analyzing the distribution, this pattern can be detected. To detect the single-threaded code anti-pattern the amount of active threads have to be analyzed. For the Pipe and Filter Architecture it is important to separate the information for each filter and monitor the execution of the individual filters.

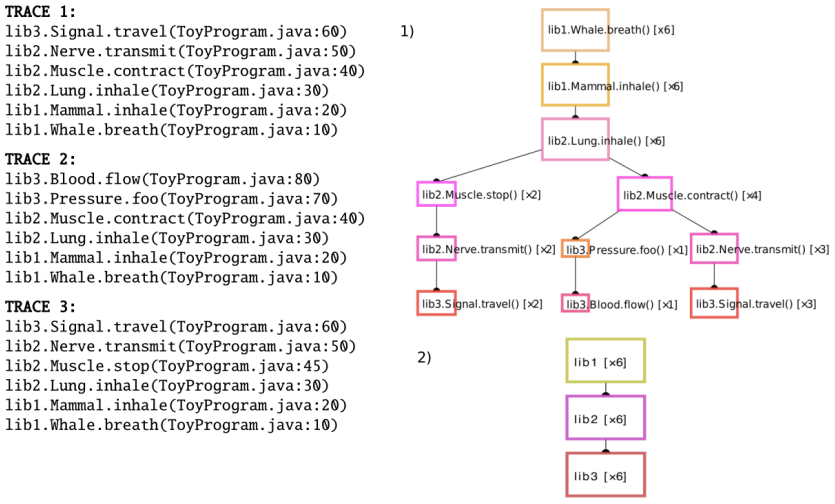


Figure 3.5.: 1) expanded call graph 2) low level abstract level [LTOB10b]

The simple visualization technique is to represent each CPU core as line in a line chart. But if we use the stack graph, the graph represents all the cores utilization and the complete CPU utilization [JME10]. Because each time series in the graph is drawn sequentially, and one time series uses the value of the previous series as a baseline and the sum of each core is the complete CPU. To prevent the case where single-threaded code is the root cause we represent the amount of running thread in a separate graph.

If the Pipe and Filter Architecture is the root cause, we need a technique or techniques which can represent the structure to visualize the Pipe and Filter architecture and can monitor each filter execution behavior separated. Lin et al. [LTOB10c] have a method to exploits the structural information present in profiling call trees to selectively raise or lower the local abstraction level of the performance data. Shown in Figure 3.5, the approach generate a call graph based on execution traces. The nodes can be compacted to reduce the amount of shown nodes. Furthermore, the size of the nodes represents the amount of executions. Due to one filter is mostly the bottleneck, we can use the color or the size for visualize the execution time. Therefore, the user gets shown the structure, the separate execution behavior and with interaction different abstract levels.

3.7. CPU-intensive Application

The CPU-intensive Application anti-pattern groups all anti-patterns with a have high CPU utilization as result. All this anti-patterns are resulting from an implementation failure, e.g., like excessive allocation, not required operations or insufficient caching.

To detect these anti-patterns human interaction is required. Measurements can be applied to identify code fragments with high CPU usage, but for the root cause extraction humans are necessary. An exception is the Insufficient Caching Introducing anti-pattern. It can be detected by identifying repeated, CPU-intensive methods which produce unique results for given inputs [Wer13].

It is obvious that we have to visualize firstly the CPU utilization. The visualization technique for CPU usage should be, like already mentioned, a line chart. The analyzed code fragments can be selected and information like CPU utilization are displayed. To reduce the effort to find the fragment in the code, we also displayed the code fragment as text (Section 2.3.5).

If the cause is Insufficient Caching it can be helpful to show the in and outputs of the detected method. The data type is Two-Dimensional with one independent variable (input) and one dependent variable (output). To visualize the data, we expand the variable structure to a vector. Each vector represent one input and each index represents one possible method output. Due to the vector structure, we can map the records on a matrix and visualize it in a matrix view [HBO10]. Shown in Figure 3.6, the y axis represents the outputs, the x axis represents the inputs and we can use the color the visualize the amount of calls with the same in and outputs. If it is Insufficient Caching the matrix looks random or has vertical lines, that means that the method returns different outputs for the same or different inputs. But if there are horizontal lines or colored dots, that means that caching can be a good option.

3.8. Database Congestion

The Database Congestion summarized all performance problems of communication and usage of databases. In case of N+1, the Stifle or Circuitous Treasure Hunt, an application performs an unnecessary high amount of database calls. The cause of the Expensive Database Call anti-pattern is on query which is unnecessarily complex or returns not entirely used data.

To detect these anti-patterns with high amount of database calls, only the analysis of a single trace is sufficient. It has to be checked if one database statement is followed by many similar statements. In case one statement is the cause, manual analysis is the only way of detecting these anti-patterns. Measurements are required to decide whether a corresponding query has a negative impact on performance [Wer13].

First of all, we have to visualize the trace, in detail the database statements. Traces are typically hierarchy data (Section 2.3.4), with parents and child as functions, and links as calls. To allow efficient interaction for exploring the trace, we will use basic indented

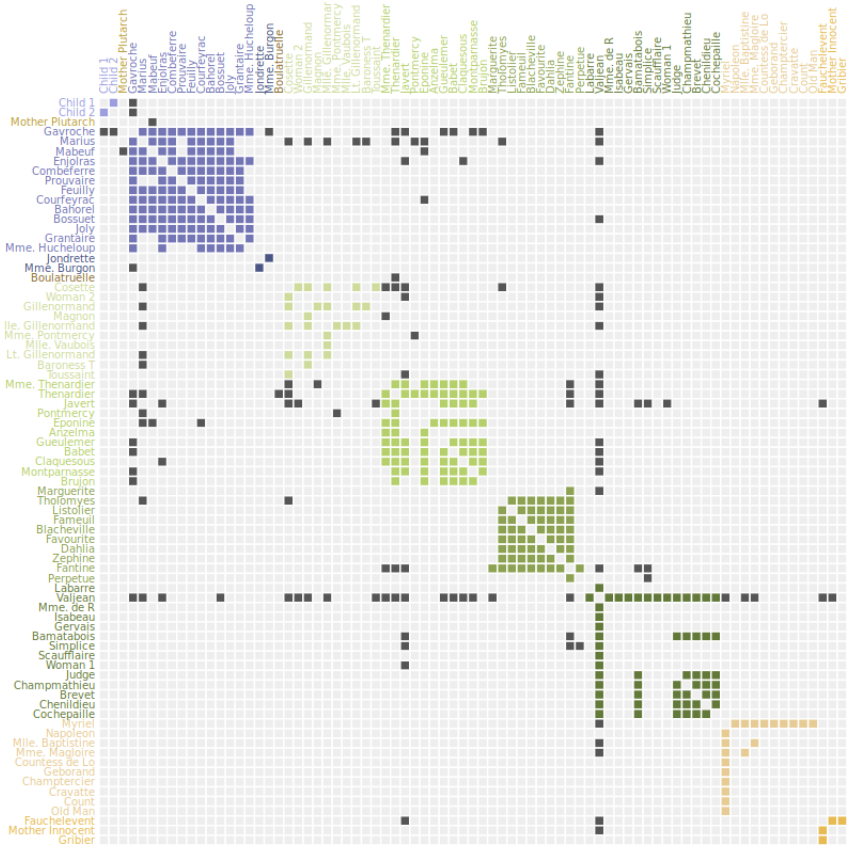


Figure 3.6.: Matrix View [HBO10]

tree layout (often used in operation system to represent file directories). A further advantage is we can already expand nodes, which directs to the databases calls.

Another approach is to use sankey diagrams [RB10] to visualize all database statements in ones. As shown in Figure 3.7, we have node and links. In this approach the nodes are the key words (*SELECT*, *FROM*, *WHERE* and *GROUP BY*) and the links are the expressions. For example in the SQL statment "*SELECT name FROM student WHERE id = '1';*" *SELECT*, *FROM* and *WHERE* are the nodes and *name*, *student* and *id = '1'* are the links. An extension would be a zoom for the links to add more nodes like *AND*, *OR* or =. Through this visualization type the user gets a good overview of all statements and sees similar statements due to the thickness of the links.

Flamegraphs [Gre16] are also a good technique for visualizing this types of anti-patterns. In case of N+1, a bunch of spikes, that are all about the same height, span over the graph.

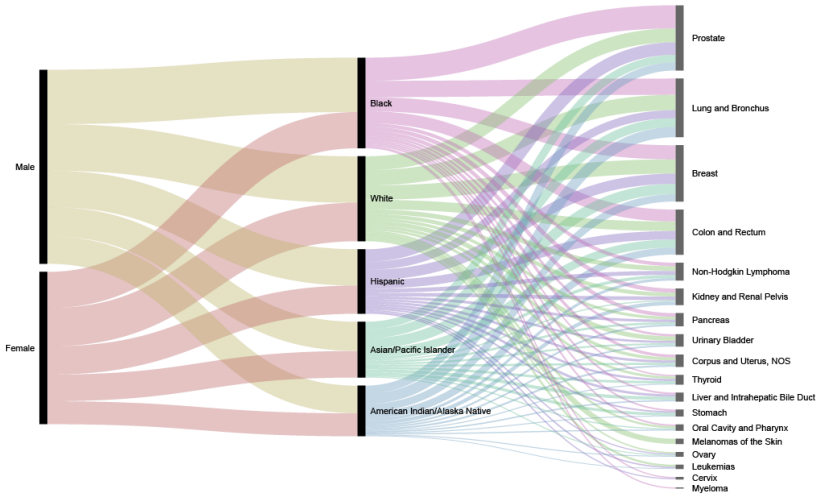


Figure 3.7.: Sankey Diagram [Med14]

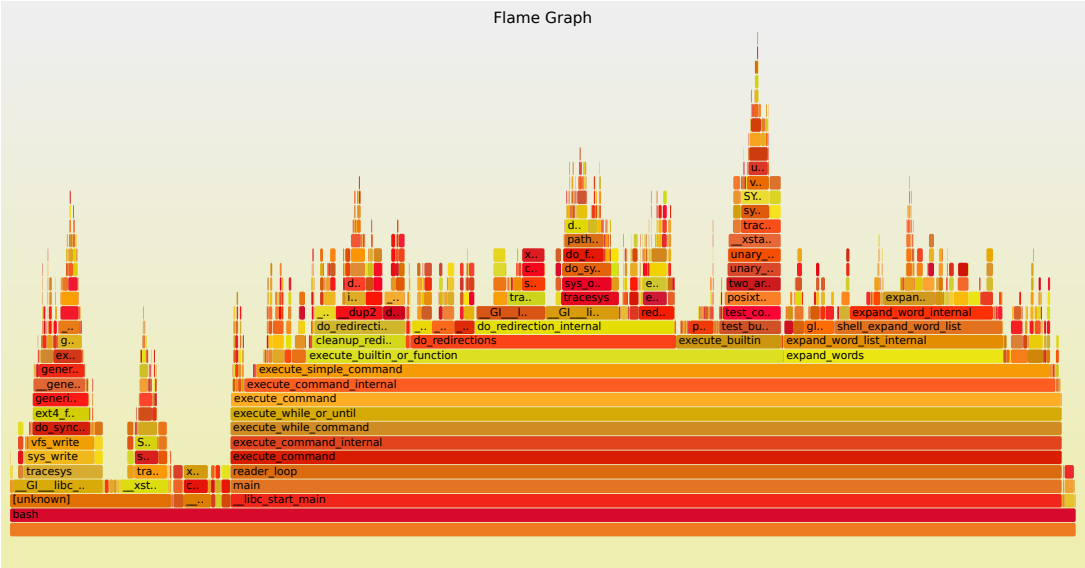


Figure 3.8.: Flamegraph [Gre16]

3.9. One Lane Bridge

The symptoms for the One Lane Bridge anti-pattern are similar to the Traffic Jam anti-pattern (Section 3.5). Causes are synchronization implementation failures, database looking or services are bottlenecks.

The detection is similar to the Traffic Jam, by conducting measurements. The coefficient of variation of response time is the key factor. The exact bottleneck, like method or service, can be detected by analyzing the relationship between load and response times.

Because the One Lane Bridge anti-pattern is similar to the Traffic Jam, the same line chart can be used to visualize the main symptom. Equivalent to the Single User Problem 3.3, *EXTRAVIS* is an option to visualize calls between service or method and their response times. This technique can cover two causes. The first one is that the service or method has too high response time. This is visible on the color of the edges. The second cause is that the load on the single service or method is too high and this is displayed by means of the amount of incoming edges.

If the root cause is the Dispensable Synchronization Dispensable anti-pattern [Wer13], we can use the *SyncTrace* [KTD13] visualization design. It shows us threads and the relationships between each other. They use attribute mapping to colors and shape of the edge bundles to encode important runtime meta-data [KTD13]. Therefore, the user can see where threads need to wait because of unnecessarily long locking areas.

Similar to the Database Congestion anti-patterns 3.8, Flamegraphs are an option to visualize bottlenecks. Because the x-axis represents time, the part where the SUT getting bogged down is presented clearly. The widest layers take the longest to run.

3.10. Excessive Messaging

Excessive Messaging occurs when the SUT communicated inefficient between components and system nodes. The Blob and the Empty Semi Truck are two causes. The Empty Semi Truck describes an unnecessarily high amount of small messages. The Blob is a single class either performs all of the work or holds all data [Smi02].

To detect the Blob static code analysis has to detect the single god class and measurements have to analyze extreme message traffic between the single class and other classes. The Empty Semi Truck detection strategy uses the message size and the payload size to analyzing the messages.

To visualize messages, we can use approaches from Call Graph Visualization techniques (Section 2.4.1). Traces can be mapped to Graphs data type (Section 2.3.6) and visualized in the common techniques with nodes and edges. The nodes represent the components or classes and the edges the messages. Therefore, the amount of edges represents the amount of calls. The extension would be *SynchroVis* (Section 2.4.1) to visualize the code structure. This allows the user to see single god classes and the excessive messaging to the class. An abstraction of *SynchroVis* is a graph with nodes as classes and the number of instances is represent by the size of the node.

Chapter 4

Implemented Visualization Tool

Our prototype, called VoP (Visualization of Performance), implement the above described visualization techniques and the integration in the application performance management (APM) life cycle. We implemented, in this thesis, the front-end part and using therefore D3.js [Mik]. D3 is a JavaScript library for manipulating documents based on data. It combines powerful visualization and interaction techniques with a data-driven approach to Document Object Model (DOM) manipulation. We integrated this in Angular [Goo]. Angular is a web application platform and combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Moreover, it supports npm as JavaScript package manager and we could integrated D3 and D3 plugins easily. In the following, we describe in Section 4.1 the general functions, in Section 4.2 the individual visualization techniques in detail and in the last Section 4.3 the possible integration in APM environments.

4.1. Features

Independent to the individual visualization techniques the prototype has overarching features. Each detected anti-pattern result is visualized in a separated tab. The techniques are independent to the anti-pattern and can reused in other anti-patterns. Due to the loose coupling, the anti-pattern representation can be changed without additional expense. The layout of the visualized report is stored in JSON format, example see in Listing 4.1. In the same way we stored the chart layout, see in Listing 4.2. Due to that, the size, the arrangement, and the data is easily and individual customizable.

Listing 4.1 Example JSON for tab layout

```
{
"name": "CPU-intensive Application",
"cols": 2,
"rowHeight": 450,
"charts": []
}
```

Listing 4.2 Example JSON for chart layout

```
{
"name": "sankey",
"data": "../assets/database-congestion/sankey.json",
"height": 350,
"width": 1200,
"colspan": 2,
"rowspan": 1,
"optional": null
}
```

4.2. Visualization Techniques

For the visualization techniques we followed some interaction rules, like *Overview and Detail* or *Extraction and Comparison* [CMS09]. Foundations, like tool-tips and labeled axis, would also be considered [PD10]. To give the user a good feel and look we include new trends [ZAL14] and paid attention to color perception. For each implemented visualization technique we provide a description and describe the particularities.

Adjacency Matrix

The adjacency matrix ($m \times n$) visualizes inputs (n) and outputs (m) of components, e.g. methods. The x axis (j) represents the inputs and the y axis (i) the outputs. A field (a_{ij}) is filled, if in the monitored time the component j had i as output. The color of the field represents the amount of appears (green=rare, red=frequent).

Line Chart

The line chart is the most used visualization technique in our prototype anti-pattern layouts. It visualizes data with a timestamp and a value. The value is represented on

the y axis and the date on the x axis. We can configure the time format, to visualize short time intervals (seconds) or longer (hours). If required, the user can view details by hovering the mouse over the line. The y axis is labeled, if only one dataset is presented, otherwise the user can see the units in the tooltip.

Spiral Chart

The spiral chart is equal to the line chart. The difference is that x axis is circular and we used bars instance of lines. In order to get the benefits, the cycle length for each cycle must be the same long. Only than can the user detect patterns. The bar hight and color is dependent to the value, higher and redder means a higher value.

Stacked Chart

The stacked chart has the same functionality than the line chart. The only different is that each time series in the graph is drawn sequentially, and one time series uses the value of the previous series as a baseline.

Call Graph

The call graph is implemented as hierarchical data. Each node has name, value and children. The value represents the amount of calls or the duration time and is visualized as size of the node or the color. Furthermore, all nodes are linked with their children nodes.

Chord

In chord we also use the D3 hierarchical data type to get the functionality of nodes and links. The generate a root node in central of the circle and adding all components as children. After this, we create out of the data links between the nodes. The links can represent a second variable, e.g. response time between components, with their style. Moreover, to get details about incoming and outgoing links, the user has to hover over the component names. The incoming links change the color to green and the outgoings to red.

Flame Graph

For the flame graph we use the *d3-flame-graph* plugin [Spi]. It needs hierarchical data as input and generates a flame graph with functions like filtering and details on demand.

Sankey

The sankey visualization in our prototype database management statements. Each node represents one key word (*WHERE, FROM, SELECT,* ...). The width of the links presents the amount of recurring parts.

Sunburst, Icicle Chart and Indented Tree

Sunbursts, icicle charts and indented trees are visualizing all the same type of data. They visualizing execution trace with different techniques. Indented trees can expand and collapse independently parts. In contrast to this, sunburst an icicle charts can only focus isolated. However, they can visualize an other variable with the width of the boxes. The difference between the sunburst and the icicle chart is that the sunburst using polar coordinates.

Horizon Chart

For the horizon chart we are using the *d3-horizon-chart* plugin [Kir]. The input data are standard time series and colors or heights are configurable.

Graph

For the graph visualization we are using force-directed graphs to reduce crossing edges. The color and the size of the nodes are variable and are depending on a variable in the record.

Code

To visualize code snippets, we are using *angular2-highlight-js* [Jon]. Therefore, we can highlighting code snippets and supporting numerous languages.

4.3. Integration

The integration in a APM environment is given by a open interface. The layouts of the anti-pattern reports and the data are stored in JSON format and can be received by a REST interface. To receive time-series data from databases we implemented a proof of concept connection via the JavaScript InfluxDB client [Ben]. InfluxDB is a time series database management system [Inf].

Chapter 5

Evaluation

We conducted a small user study to evaluate the quality of the visualized report, which our prototype reports tool generated. With regard to root cause detection and performance problem understanding, we add a task to analyzing the usefulness.

5.1. Evaluation Goal

The research question we seek to answer is *How visualized anti-pattern detection report are perceived by practitioners?* and *Can the report contribute to better anti-pattern and root causes understanding?*. Furthermore, we want to evaluate the individual visualization techniques and find out which technique is better, for given anti-pattern and data, than other.

5.2. Evaluation Design

We designed a user study composed in three parts. The first part is about the personal experience and background about anti-patterns. The second is a task to evaluate the report functionality. The last part is a survey to evaluate the usability and the single visualization techniques. The question we ask in the first part, covers the level of expertise in anti-pattern, performance problems detection and human computer interaction areas. The task, the participants have to solve, is to detect the anti-patterns based only on the presented data. Each of the ten anti-patterns are visualized separated and use different visualization techniques (see Appendix A for example reports). The participants receive the possible anti-patterns, with a short description, in advance. This part is necessary to analyze the usefulness of the visualization techniques combinations.

		Experience		
Participants	**Education**	**Anti-pattern**	**PPD**	**HCI**
P1	Master Student	Intermediate	Beginner	Intermediate
P2	Master Student	Beginner	Intermediate	Intermediate
P3	Bachelor Student	Intermediate	Beginner	Expert
P4	Bachelor Student	Beginner	Beginner	Intermediate
P5	Industry	No knowledge	Beginner	Beginner
P6	Master Student	Beginner	Intermediate	Intermediate
P7	Industry	Beginner	No knowledge	Beginner
P8	Bachelor Student	Beginner	Beginner	Expert
P9	Bachelor Student	Beginner	Beginner	Intermediate

Table 5.1.: User study participants (PPD = Performance Problem Detection, HCI = Human Computer Interaction)

The performance data and the combinations are manually created and not measured on existing systems. But the data represent the behavior of real systems with performance problems. The third part poses a set of questions about the report content, the interaction, preferred techniques and improvement suggestions (see Appendix B for question form). All participants received the same informations about anti-patterns and the prototype.

5.3. Results

Figure 5.1 shows the results of the task. The x axis groups the tabs and the y axis represents the amount of answers. The correct answers are in the legend order (*One Lane Bridge = Tab1*, *Application Hiccups = Tab2*). In the most cases the highest bar is the correct answer, e.g. *Tab2* every participants had the correct solution. For example, the cause for the variation in *Tab9* is that *Traffic Jam* is a symptom from other anti-patterns and is sometimes difficult to speculate more precisely. Although the most participants are beginner in the anti-pattern area and only had a short descriptions of the anti-pattern behaviors, the results are positive. Moreover, we asked which visualization technique was most useful for each anti-pattern. Unfortunately, based on the result we can not say that some techniques are better than any other.

The results of the feedback indicate that all participants agreed that visualization reports will help to understand performance problems. They also agreed that visualization reports help to find root causes.

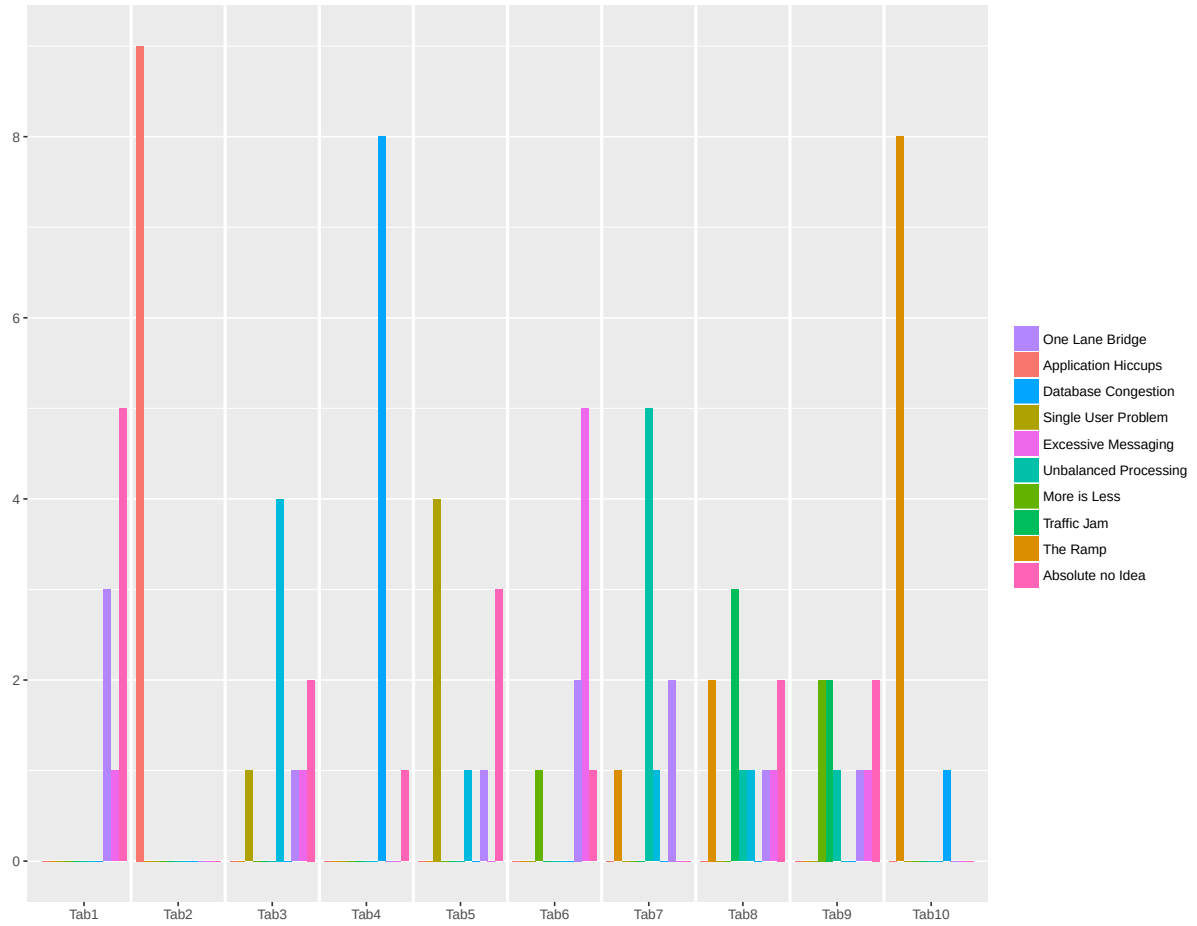


Figure 5.1.: Evaluation task results

(a) Question: Do you think that this visualization report will help you to understand performance problems?

(b) Question: Do you think that this visualization report will help you to find easier root causes?

Figure 5.2.: Evaluation question results

All participants gave positive remarks on using the visualization report. They gave use some improvements, like P8 suggested that adding a overarching interaction feature to zoom out and in all data at once can improve the root cause detection. P9 said better axis labeling can help to understand measurements. Participant P2 liked the tooltips, but self-explanatory views would be preferable.

Chapter 6

Conclusion

In this chapter, we summarize the work we did throughout this thesis, discuss it further and recommend the work for the future.

6.1. Summary

In this work, we presented a approach to visualize performance anti-pattern detection results. The approach is based on existing visualization techniques and connect this with APM tools measurements. The main part deals with analyzing anti-pattern and finding visualization techniques to improve the performance problem understanding and root cause detection. Furthermore, we considered data types to map the monitored measurements on potential techniques. To evaluate the approach we implemented the prototype *VoP*. It can handle different data types and visualize them in different visualization techniques.

Finally, we evaluated the implemented prototype and the potential anti-pattern result visualizations. We used a user study to receive feedback and evaluate the usefulness. The evaluation showed that performance problem comprehensibility and the root cause isolation are improved. Moreover, the improvement possibilities of the prototype are presented.

6.2. Discussion

In Section 1.2, we described the goals for this thesis. Now we will discuss whether we have achieved the goals.

The first goal was to find and categorize visualization techniques. We have decided that we categorize the techniques based on the goals of the performance visualization technique. We described many different communed and more specialized techniques and have successfully found categorizes in Section 2.4.

The second goal was to find a mapping between anti-patterns and visualization techniques. For some anti-patterns it was easier to find techniques, which visualized the anti-pattern behavior. On the other hand, some anti-pattern need more specific knowledge about the system and for this reason the detection result can not be visualized that easy.

The third goal was to implement a prototype. We have implemented a prototype, called *VoP*, which can visualize anti-pattern detection results. It also supports a variety of visualization techniques. The opportunity to receive measurements data from APM tools was considered, but not completely implemented.

The last goal of this thesis was the evaluation. A user study was successfully executed. Both the prototype and the report layouts were evaluated. The feedback was mostly positive and the participants had good improvement ideas.

6.3. Future Work

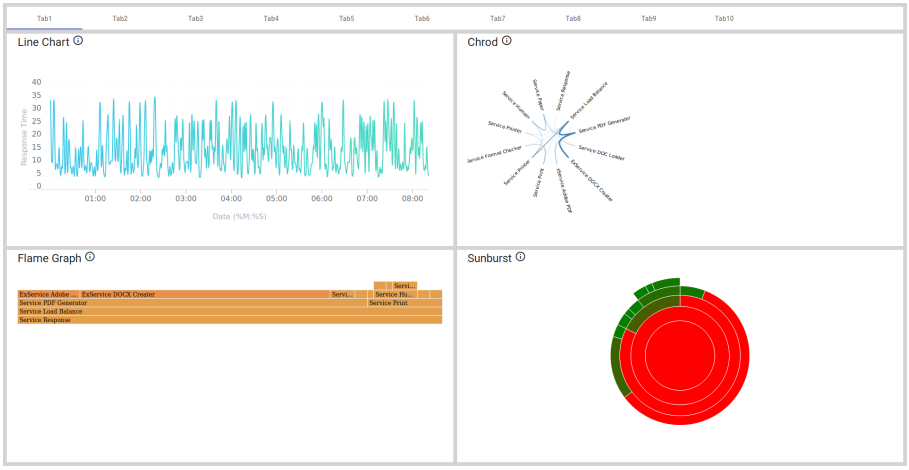
In this section, we discuss potential enhancements of our work.

The first recommended work is to connect existing APM tools with our prototype. Furthermore, the functionality and the variety of visualization techniques can be extended. To improve the details on demand, overreaching filtering can be integrated to focus in larger software system on performance weak components. To guarantee the independence to APM tools, (OPEN.XTRACE) and a interface to communed time series database management systems should be integrated. Visualization techniques, like SyncTrace or SynchroVis are good approaches, but the implementation in the prototype exceed our possibilities.

The literature can also be researched further for visualization techniques. New approaches are constantly emerging, which facilitate the understanding of performance data.

Appendix A

VoP Example



(a) Tab 1

(b) Tab 2

Figure A.1.: VoP Example

Appendix B

Evaluation Form

General Questions

1. In which environment do you work?
   - ○ Bachelor Student
   - ○ Master Student
   - ○ University
   - ○ Industry
   - ○ Other
2. What is your level of expertise in following area?

	No knowledge	Beginner	Intermediate	Expert
Anti-pattern	○	○	○	○
Performance Problems Detection	○	○	○	○
Human Computer Interaction	○	○	○	○

Task

Now we have a task for you. Please visit this web page: http://62.75.211.42:4200/ You can see 10 anti-pattern detection reports (everyone in a separated tab). Each report visualize one anti-pattern and in the following we will ask you which tab visualize which anti-pattern. If your a beginner in anti-patterns or need a refresher feel free to take a look in the anti-pattern description: http://euve258746.serverprofi24.de/index.php/s/mQwb5ixwQitD1D2

4. Which anti-pattern is visualized by Tab *i*? *(i from 1 to 10)*

5. And which visualization technique helped you the most in Tab *i*? *(i from 1 to 10)*

Feedback Questions

6. Do you think that this visualization report will help you to understand performance problems?

Strongly Disagree ○ ○ ○ ○ ○ Strongly Agree

7. Do you think that this visualization report will help you to find easier root causes?

Strongly Disagree ○ ○ ○ ○ ○ Strongly Agree

8. Did you need more hints to understand the visualization techniques?

9. What kind of interaction do you missed?

10. Did you missed some features or have some improvements?

Appendix B

Bibliography

[ABF+10] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N. R. Tallent. “HPCTOOLKIT: Tools for performance analysis of optimized parallel programs.” In: *Concurrency Computation Practice and Experience* 22.6 (Apr. 2010), pp. 685–701 (cit. on p. 13).

[AH10] A. Adamoli, M. Hauswirth. “Trevis.” In: *Proceedings of the 5th international symposium on Software visualization - SOFTVIS ’10* (2010), p. 73 (cit. on p. 13).

[AKG10] E. Aftandilian, S. Kelley, C. Gramazio. “Heapviz: interactive heap visualization for program understanding and debugging.” In: *Software visualization* (2010), pp. 53–62 (cit. on p. 12).

[Ben] C. P. Ben Evans. *node-influx | An InfluxDB Client for JavaScript*. URL: https://node-influx.github.io/ (visited on 12/24/2017) (cit. on p. 35).

[CFA+06] A. M. Cheadle, A. J. Field, J. W. Ayres, N. Dunn, R. A. Hayden, J. Nystrom-Persson. “Visualising dynamic memory allocators.” In: *Proceedings of the 2006 international symposium on Memory management - ISMM ’06*. New York, New York, USA: ACM Press, 2006, p. 115 (cit. on p. 15).

[CHZ+07] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. Van Wijk, A. Van Deursen. “Understanding execution traces using massive sequence and circular bundle views.” In: *IEEE International Conference on Program Comprehension*. 2007, pp. 49–58 (cit. on pp. 12, 22).

[CMS09] S. K. Card, J. D. Mackinlay, B. Shneiderman. “Information Visualization.” In: *Human-computer interaction: Design issues, solutions, and applications* 181 (2009) (cit. on p. 32).

[CR11] A. N. M. I. Choudhury, P. Rosen. “Abstract visualization of runtime memory behavior.” In: *Proceedings of VISSOFT 2011 - 6th IEEE International Workshop on Visualizing Software for Understanding and Analysis*. IEEE, Sept. 2011, pp. 1–8 (cit. on p. 15).

[Dat] Datadog. *Datadog - Modern monitoring & analytics*. URL: https://www.datadoghq.com/ (visited on 12/27/2017) (cit. on p. 17).

[Döh12] P. Döhring. “Visualisierung von Synchronisationspunkten in Kombination mit der Statik und Dynamik eines Softwaresystems.” Master thesis. Kiel University, 2012 (cit. on p. 14).

[DWB13] W. De Pauw, J. Wolf, A. Balmin. “Visualizing jobs with shared resources in distributed environments.” In: *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013*. IEEE, Sept. 2013, pp. 1–10 (cit. on pp. 15, 16).

[ES92] S. G. Eick, J. L. Steffen. “Visualizing code profiling line oriented statistics.” In: *IEEE Conference on Visualization*. IEEE Comput. Soc. Press, 1992, pp. 210–217 (cit. on p. 13).

[Gal] D. Galles. *Data Structure Visualizations*. URL: https://www.cs.usfca.edu/%7B~%7Dgalles/visualization/Algorithms.html (cit. on p. 12).

[Goo] Google ©2010-2017. *Angular*. URL: https://angular.io/ (visited on 12/22/2017) (cit. on p. 31).

[Gra] Grafana Labs. *Grafana - The open platform for analytics and monitoring*. URL: https://grafana.com/ (visited on 12/27/2017) (cit. on p. 17).

[Gre16] B. Gregg. “The flame graph.” In: *Communications of the ACM* 59.6 (2016), pp. 48–57 (cit. on pp. 13, 27, 28).

[HBO10] J. Heer, M. Bostock, V. Ogievetsky. “A tour through the visualization zoo.” In: *Communications of the ACM* 53.6 (June 2010), p. 59 (cit. on pp. 14, 24, 26, 27).

[HHMO17] C. Heger, A. van Hoorn, M. Mann, D. Okanović. “Application Performance Management : State of the Art and Challenges for the Future.” In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*. ACM, 2017, pp. 429–432 (cit. on pp. 1, 2, 5).

[HHO+16] C. Heger, A. van Hoorn, D. Okanović, S. Siegl, A. Wert. “Expert-Guided Automatic Diagnosis of Performance Problems in Enterprise Applications.” In: *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 185–188 (cit. on p. 5).

[HKA09] J. Heer, N. Kong, M. Agrawala. “Sizing the horizon.” In: *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. 2009, p. 1303 (cit. on p. 15).

[HN16] A. Hidiroglu, NovaTec Consulting GmbH. “Detecting Performance Anti-Patterns in Enterprise Applications by Analyzing Execution Traces Foundations and state of the art.” In: (2016), pp. 1–21 (cit. on pp. 21, 23).

[IGJG14] K. Isaacs, A. Giménez, I. Jusufi, T. Gamblin. “State of the Art of Performance Visualization.” In: *Proc. of Eurographics on Visualization (EuroVis)* (2014) (cit. on pp. 2, 11, 15).

[Inf] Influxdata. *InfluxDB*. URL: https://www.influxdata.com/time-series-%20platform/influxdb/ (visited on 12/24/2017) (cit. on p. 35).

[JME10] W. Javed, B. McDonnel, N. Elmqvist. “Graphical perception of multiple time series.” In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (2010), pp. 927–934 (cit. on pp. 22, 23, 25).

[Jon] Jonathan Chase. *angular2-highlight-js*. URL: https://github.com/jaychase/angular2-highlight-js (visited on 12/23/2017) (cit. on p. 34).

[Kei02] D. A. Keim. “Information visualization and visual data mining.” In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 1–8 (cit. on p. 10).

[Kir] Kiril Mandov. *D3 Horizon Chart*. URL: http://kmandov.github.io/d3-horizon-chart/ (visited on 12/23/2017) (cit. on p. 34).

[KTD13] B. Karran, J. Trümper, J. Döllner. “SYNCTRACE: Visual thread-interplay analysis.” In: *2013 1st IEEE Working Conference on Software Visualization - Proceedings of VISSOFT 2013* (2013) (cit. on pp. 16, 29).

[LLB+12] A. G. Landge, J. A. Levine, A. Bhatele, K. E. Isaacs, T. Gamblin, M. Schulz, S. H. Langer, P. T. Bremer, V. Pascucci. “Visualizing network traffic to understand the performance of massively parallel simulations.” In: *IEEE Transactions on Visualization and Computer Graphics* 18.12 (Dec. 2012), pp. 2467–2476 (cit. on p. 14).

[LTOB10a] S. Lin, F. Taïani, T. C. Ormerod, L. J. Ball. “Towards anomaly comprehension: using structural compression to navigate profiling call-trees.” In: *Proceedings of the 5th international symposium on Software visualization*. Section 7. 2010, pp. 103–112 (cit. on p. 13).

[LTOB10b] S. Lin, F. Taïani, T. C. Ormerod, L. J. Ball. “Towards anomaly comprehension: using structural compression to navigate profiling call-trees.” In: *Proceedings of the 5th international symposium on Software visualization* Section 7 (2010), pp. 103–112 (cit. on p. 25).

[LTOB10c] S. Lin, F. Taïani, T. C. Ormerod, L. J. Ball. “Towards anomaly comprehension: using structural compression to navigate profiling call-trees.” In: *Proceedings of the 5th international symposium on Software visualization*. Section 7. 2010, pp. 103–112 (cit. on p. 25).

[Med14] D. S. Media. *Visualizing Categorical Data as Flows with Alluvial Diagrams | Digital Splash Media*. 2014. URL: http://digitalsplashmedia.com/2014/06/visualizing-categorical-data-as-flows-with-alluvial-diagrams/ (visited on 12/14/2017) (cit. on p. 28).

[Mik] Mike Bostock. *D3.js - Data-Driven Documents*. URL: https://d3js.org/ (visited on 12/22/2017) (cit. on p. 31).

[OHH+16] D. Okanović, A. van Hoorn, C. Heger, A. Wert, S. Siegl. “Towards performance tooling interoperability: An open format for representing execution traces.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9951 LNCS. Springer, Cham, Oct. 2016, pp. 94–108 (cit. on p. 6).

[OJHS04] A. Orso, J. Jones, M. J. Harrold, J. Stasko. “GAMMATELLA: visualization of program-execution data for deployed software.” In: *Proceedings of the 1st ACM symposium on Software visuallization*. 2004, pp. 699–700 (cit. on p. 13).

[Par07] T. Parsons. “Automatic detection of performance design and deployment antipatterns in component based enterprise systems.” PhD thesis. 2007 (cit. on p. 6).

[PD10] B. Preim, R. Dachselt. *Interaktive Systeme: Band 1: Grundlagen, Graphical User Interfaces, Informationsvisualisierung*. Springer-Verlag, 2010 (cit. on pp. 19, 32).

[PHD10] W. D. Pauw, S. Heisig, W. De Pauw. “Zinsight: a visual and analytic environment for exploring large event traces.” In: *Proceedings of the 5th international symposium on Software visualization*. 2010, pp. 143–152 (cit. on p. 12).

[Pla86] W. Playfair. *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England during the Whole of the Eighteenth Century*. 1786 (cit. on p. 15).

[RB10] M. Rosvall, C. T. Bergstrom. “Mapping change in large networks.” In: *PLoS ONE* 5.1 (2010) (cit. on p. 27).

[Shn96] B. Shneiderman. “The eyes have it: a task by data type taxonomy for information visualizations.” In: *Proceedings 1996 IEEE Symposium on Visual Languages*. 1996, pp. 336–343 (cit. on p. 10).

[Smi02] L. G. Smith, Connie U and Williams. “New Software Performance AntiPatterns: EvenMore Ways to Shoot Yourself in the Foot.” In: *Computer Measurement Group Conference* (2002), pp. 667–674 (cit. on pp. 8, 29).

[SMM13] C. Sigovan, C. W. Muelder, K. L. Ma. “Visualizing large-scale parallel communication traces using a particle animation technique.” In: *Computer Graphics Forum* 32.3 PART2 (June 2013), pp. 141–150 (cit. on p. 16).

[Spi] M. Spier. *d3-flame-graph*. URL: https://github.com/spiermar/d3-flame-graph (visited on 12/23/2017) (cit. on p. 34).

[SW00] C. U. Smith, L. G. Williams. “Software Performance AntiPatterns.” In: *Computer Measurement Group Conference* December (2000), pp. 717–725 (cit. on p. 9).

[SW02] C. Smith, L. Williams. “Software Performance AntiPatterns; Common Performance Problems and their Solutions.” In: *Cmg-Conference-* 2 (2002), pp. 797–806 (cit. on p. 8).

[SW03] C. U. Smith, L. G. Williams. “More New Software Performance AntiPatterns: EvenMore Ways to Shoot Yourself in the Foot.” In: *Computer Measurement Group Conference* (2003), pp. 717–725 (cit. on p. 9).

[TS08] C. Tominski, H. Schumann. “Enhanced Interactive Spiral Display.” In: *The Annual SIGRAD Conference Special Theme: Interaction*. 2008, pp. 53–56 (cit. on p. 21).

[Wer13] A. Wert. “Performance problem diagnostics by systematic experimentation.” In: *Proceedings of the 18th international doctoral symposium on Components and architecture - WCOP ’13*. WCOP ’13. New York, NY, USA: ACM, 2013, p. 1 (cit. on pp. 6–9, 20, 24, 26, 29).

[WGK10] M. O. Ward, G. Grinstein, D. Keim. *Interactive data visualization: foundations, techniques, and applications*. CRC Press, 2010 (cit. on pp. 10, 11).

[WH96] M. W. Wolfgang E. Nagel, Alfred Arnold, K. S. Hans-Christian Hoppe. “VAMPIR: Visualization and analysis of MPI resources.” In: *Supercomputer 63* XII (1996), pp. 69–80 (cit. on p. 17).

[WKK16] J. Walter, M. König, S. Kounev. “PAVO: A Framework for the Visualization of Performance Analyses Results.” In: (2016) (cit. on p. 17).

[WL07] R. Wettel, M. Lanza. “Visualizing Software Systems as Cities.” In: (2007), pp. 92–99 (cit. on p. 14).

[WL90] S. Wehrend, C. Lewis. “A Problem-oriented Classification of Visualization Techniques.” In: *Proceedings of the 1st Conference on Visualization ’90*. VIS ’90. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, pp. 139–143 (cit. on p. 11).

[WWF+13] J. Waller, C. Wulf, F. Fittkau, P. Dohring, W. Hasselbring. “Synchrovis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency.” In: *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, Sept. 2013, pp. 1–4 (cit. on p. 14).

[WYH10] Y. Wu, R. Yap, F. Halim. “Visualizing Windows system traces.” In: *Proceedings of the 5th international symposium on Software visualization*. 2010, pp. 123–132 (cit. on pp. 13, 22).

[ZAL14] E. Zudilova-Seinstra, T. Adriaansen, R. van Liere. *Trends in Interactive Visualization: State-of-the-Art Survey*. Springer Publishing Company, Incorporated, 2014 (cit. on p. 32).

All links were last followed on January 02, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature