Information visualization of microservice architecture relations and system monitoring

A case study on the microservices of a digital rights management company - an observability perspective

MARCUS FRISELL
Information visualization of microservice architecture relations and system monitoring
A case study on the microservices of a digital rights management company
- an observability perspective

Marcus Frisell
Royal Institute of Technology
Stockholm, Sweden
mfrisell@kth.se

ABSTRACT

90% of the data that exists today has been created over the last two years only. Part of the data space is created and collected by machines, sending logs of internal measurements to be analyzed and used to evaluate service incidents. However, efficiently comprehending datasets requires more than just access to data, as Richard Hamming puts it; “The purpose of computing is insight, not numbers.” A tool to simplify comprehension of complex datasets is information visualization, which works by transforming layers of information into a visual medium, enabling the human perception to quickly extract valuable information and recognize patterns.

This was an experimental design-oriented research study, set out to explore if an information visualization of microservice architecture relations combined with system health data could help developers at a Swedish digital rights management company (DRMC) to find root cause incidents, increase observability and decision support, i.e. simplifying the incident handling process.

To explore this, a prototype was developed and user tests consisting of a set of tasks as well as a semi-structured interview was executed by ten developers at DRMC. The results concluded that the proposed solution provided a welcomed overview of service health and dependencies but that it lacked the ability to effectively focus on certain services, essentially making it difficult to find root causes.

Visualizations like this seems to be best suited for overview-, rather than focused, comprehension. Further research could be conducted on how to efficiently render large complex datasets while maintaining focus and how to account for external factors.

Author Keywords
Information visualization, microservices, microservice architecture, relations, system monitoring, observability, decision support, root cause, incident

ACM Classification Keywords
Information visualization, microservices, microservice architecture, relations, system monitoring, observability, decision support, root cause, incident

1. INTRODUCTION

We are living in a stream of information. We share and experience everything we, our friends, colleagues, companies and customers do. Merely ten years ago humanity as a whole could store about 290 exabytes of data [9]. Currently 2.5 exabytes, or 2.5 billion gigabytes of data is generated each day. Over the last two years only, we have created 90% of the data that exists today [34], largely due to our documentation in social media.

During the last decade we’ve been part of a societal paradigm shift brought on to us by the influence of social media and a constant influx of faster, smaller and less expensive computers and sensors. Currently over 50% of the global population has access to the internet [29], and over a quarter has an active Facebook account [12]. Measuring user data has become crucial, and is often necessary to avoid losing competitive advantage [19]. This data enables companies to draw specific and general conclusions about individuals or groups of people. For example, companies like Facebook and Google uses this data to display customized ads and to improve platform features.

While collecting user action data is important in exploring improvements to a platform, collecting data about system metrics is important to know how well the platform is functioning and performing, thus simplifying the maintenance process by increasing observability. Observability is defined as a measurement of how well a system operates depending on it’s output [13] and is important in the performance evaluation of the service.

Most organizations rely on large quantities of distributed resources to function [27], collecting data to gain insights into how these resources are behaving is the best way to measure the well being of the service. Data insights plays a critical role in the troubleshooting and the post-mortem process, which is the documentation of successful and unsuccessful project and or data elements.

Separating the back-end-platform into modular, independently deployable services is called microservice architecture and is a way of simplifying the maintenance process, as well as ease scalability and the ability to extend functionality outside of one target platform. Each microservice is often in charge of one particular process.

In applications where data revisions matter, real time- or dynamic data collection and monitoring is important [5]. Real time data is defined as data that is presented immediately when it is collected [20].

The ability to simplify the comprehension of complex datasets can be enabled by turning layers of information into another medium. By transforming numerical data points to animations or images the focus shifts from laborious searching for anomalies or interesting points of data to visual perception, and the potential to efficiently absorb information increases. To quickly extract information without having to think about it is called pre-attentive processing, and is a key to the value of information visualization [23].

Card, Mackinlay, and Shneiderman define information visualization as “the use of computer-supported, interactive visual representations of data to amplify cognition.”[3]. The first information visualizations were printed on paper, which made sight the only way to interact with them, but with computers the ability to dynamically update how data was visualized was enabled, and with a greater amount of processing power and higher resolution screens more complex data could be shown at
once, without compromising comprehension. Information visualization facilitates structuring of information in a sensible way, highlighting important points of data. Such tools excels at tasks where the user explores a dataset to gain new insights [23]. The exploration, or visual data mining, is made possible by giving the user navigation possibilities not present in, for example, numerical datasets, and can often be a better alternative than developing complex algorithms to gain data insights, by instead leveraging a users perception [14]. An effective information visualization platform will make the user reflect upon target issues like “Where am I? Where can I go? How do I get there? What lies beyond? Where can I usefully go?” as well as providing means of doing so [23].

Different datasets needs different visualizations, depending on what dimensions of data they represent. The data can be summarized in two main types; values and relations [23]. The former represents data that has a continuous attribute, such as weather temperature or CPU load, whereas relations represent data such as a parent or car owner, which is a distinct attribute.

This study will focus on the relations part of the data types.

1.1 Case Description

The tool evaluated in this study was developed in collaboration with a Swedish digital rights management company, henceforth named DRMC. DRMC caters to millions of people, and behind the company are hundreds of developers. They create features for the different applications the company provides, design user interfaces, control payment transactions and much more. Primarily, there are people building and maintaining the back-end, i.e., the data process layer and physical hardware, on which the aforementioned developers base their work. The back-end consists of thousands of microservices all running simultaneously, generating vast amounts of useful data. To ensure that nothing breaks, the data is monitored for patterns, and decisions about what could be improved for the future are made.

The main platform for system monitoring at DRMC is a dashboard showing time series of individual microservices with owner determined relevant data such as CPU usage or user connections. The dashboard will henceforth be referenced by Dash. The developers set up alerts so that when metrics meets a certain criteria they get notified, ensuring that they are aware of incidents. However, this doesn’t help them figure out where a problem originates from, only that a certain microservice is defective.

1.2 Previous Interviews

The group at DRMC responsible for continuous evaluation of services and workflows, recently conducted interviews regarding system monitoring on twenty DRMC developers. The report have served as source material to the thesis and have informed the initial formulation of the research area. It concluded that the developers wish for a tool to make it easier to find the root cause to service incidents as well as who is responsible. For example stating that the developers “do not have insight into dependencies, but they are usually a root cause for incidents[...],” that “[...]it can be a bit tricky to find out which team who own the service [...]” and that “When I see the alarm, I only see that it is a service and the name of it, and then I need to find out which team owns it, and are they having problem with it.”

Another developer states that “[...]we have problems understanding who is responsible for bad/broken/ wrong data[...] We need to troubleshoot this manually and we engage in this errands multiple times a week.” And when considering that “[...]we can take 1-2h to just talk to people about what happened last night [...]” and that “We look at graphs from other services to understand the problem.” we can see that there is great room for research and improvement.

When questioned with how to define observability a user states that it’s “[...]the possibility to have a graph with dependencies and the ability to also see the amount of affected users.”

1.3 Research Question

The thesis was a design-oriented experimental research study with the goal of finding out a way of increasing observability and decision support in the troubleshooting of microservice incidents, by developing a prototype to visualize the back-end services and test it on a set of expert users. The study aimed to further the knowledge about how to efficiently visualize microservice architecture relations and system monitoring by creating a design artifact [30].

With this in mind, a research question was formulated:

Does a information visualization of microservice architecture relations and system monitoring simplify the incident handling process?

To help answer this question two sub-questions was formulated.

- Does the prototype increase observability and decision support?
- Does the prototype help developers find root cause incidents?

1.4. Delimitations

The study aimed to solve a certain problem for a specific target group. The prototype was assessed as an expert tool intended for expert users, i.e., users that had enough knowledge to assess if the prototype was satisfactory in answering the research question.

This was not a comparative study of the currently used system monitoring system Dash, and neither was the purpose of the proposed solution meant to enhance the current platform. Instead the focus was a new approach that could reach more purposes than earlier. A single proposed solution was tested by the users.

Recently, complex visualization evaluation based on trivial metrics such as time of task completion has been criticized [17]. This study will base the evaluation of how well the prototype answered the research question on the expert users testimonies and observed interaction.

2. BACKGROUND & THEORY

This chapter will establish related work and information visualization as a tool.

2.1 Related Work

Presenting an abundance of data to a user can be intimidating and might limit their ability to absorb relevant information. A solution to this problem is following the information-seeking mantra coined by Ben Shneiderman: Overview first, zoom and filter then details on demand [21]. The idea is to present the user with all the data, then give the ability to filter and remove irrelevant data to focus on a certain part of the data space or look for interesting patterns, and lastly providing additional information about the data.

Ilinisky, N. and Steele, J states that giving elements different encodings such as color, size and position [11] is important to
distinguish them, and Ming-Kuei Hu states they play a major role in recognizing patterns [10]. Additionally, elements should be positioned closer together if they are related, according to the cluster hypothesis [4].

Relational data is often represented as flow- or tree charts, i.e. charts where objects are ordered as a connected sequence, describing a process [26]. The objects are called nodes and their relations - edges [14].

As a means of exploring possibilities for system monitoring, Dahl J. conducted a study regarding system monitoring data presentation. His solution presented the data as a time series, along with a tree map. This enabled focus on certain sections of the data, while the ability to maintain an overview remained [6]. The clustering in the proposed solution made it possible to distinguish between local incidents and global, a solution dr John Snow already established amid the Cholera outbreak in London during the 1850’s, in something that can be considered one of the earlier information visualizations. He marked reported cases of cholera and water wells on map. Visually aggregating the information, he could draw the conclusion that most of the reported cases were spread around a certain well. By shutting it down the reported cases diminished [22].

[diagram: Vizceral - an information visualization showing data flow between Netflix’s services]

Other means of quickly getting an overview of microservice health has been researched by a group lead by senior developer Justin Reynolds at Netflix. They’ve created a tool called Vizceral to visualize current incoming user connections flowing between services [33]. See figure 1. Since the amount of data is too large to effectively render 1:1, they instead utilize the tool to look for relative sizes in the flow of data, assessing if services are responding correctly. The layout and distance of the nodes depend on how important they are to each other, determined by how much data flows between them, as well as rendering minimal relational overlap. In Vizcerals detailed view, the user can focus on certain services, getting metrics and highlighting neighbors. Unhealthy services are rendered red, making them stand out [15].

Buoyant is a provider of service mesh infrastructure that together with the consulting firm Bocoup developed a tool to visualize microservice architecture. The tool presents service topology as a graph with nodes and edges whose thickness symbolizes the data flowing between them. The approach is similar to Vizceral, although a bit more minimalistic [28].

Another tool developed to visualize system metrics, or more specifically computer logs, is Tudumi. It aggregates long lists of logs into concentric disks, splitting different types of information into layers, with the most critical information on top. Summarizing logs the tool can estimate and present usual occurrences and anomalies [24].

Real time data analysis is often used in navigation, and most recently in self driving cars, were quickly responding to anomalies is very important [35]. Earlier it has been incorporated into many other sectors, such as economics, health or business intelligence. However, advancements in real time visualizations has come more recently. Gephi is a tool that in real time visualizes networks as a graph consisting of nodes and edges. It’s developed to be an efficient analysis tool to better understand complex networks with up to 20 000 nodes. By ordering and clustering nodes, users can comprehend dynamically updated changes instantly [1].

2.2 Ethical Aspects

As established, information visualization is a way of presenting and interacting with data that makes content easier to comprehend. By skewing or showing datasets in different ways, vastly different conclusions regarding the same dataset can be made. The ability to guide a user to a certain conclusion or action based on perception is called affordance [7], a term founded by psychologist JJ Gibson. The ethical aspect of general information visualization mustn’t be disregarded, however it’s relevance when evaluating the expert tool developed for this study is slim.

Collection of data poses its own set of responsibility. As stated by the GDPR (General Data Protection Regulation), collecting personal user data must be specifically declared and approved by the user [25]. Since this thesis focuses on system-, rather than personal data, the ethical aspect of data collection is diminished.

3. METHOD

This chapter will introduce the scientific methodology, i.e., the development process of the prototype with regards to the interviews conducted by DRMC and earlier established related work, the creation of the user tests and associated tasks, as well as how the collected data was categorized and prepared for analysis. This process is described in figure 2.

[diagram: The scientific process]

3.1 Earlier Interviews & Related Work

Previously mentioned in section 1.2 respectively 2.1.

3.2 Development Process

It became clear from the initial user studies conducted by DRMC that the two most important factors to visualize is owned services and their health and dependencies. The majority of the developers are responsible for only their own services therefore the visualization will present the user with an initial view of a subset of all data, namely their own, instead of presenting all available data. Presenting this already zoomed in view first hopefully gives the developers the ability to quickly find their unhealthy services and navigate further into the dataset from there.

DRMC’s back-end consists of more than four thousand microservices with a thousand different owners maintaining them. Visualizing every part of this system would be very process-intensive for the computer and more importantly
extremely confusing for the user. To make sure the application shows the right amount of relevant data, the user specifies which microservice owner (which from now on will be called core) they would like to examine. From there the application algorithmically creates a tree-like flow chart of microservice-clusters that have some relation to the specified owner. The relationship data is programmatically and manually aggregated and reached through an API. For this prototype a static dump of data is used. In accordance with the cluster hypothesis, the core is shown as a cluster of nodes in the center of the view and layers of clusters are displayed to the left (upstream) and right (downstream). The data in this “tree” flows from left to right, trying to make it more intuitive than the back and forth relations available in DRMC’s static internal system presentation.

The distance from a cluster to the center is determined by how short it’s relation to the core is, i.e. how many services data flows through before reaching the end. E.g. the clusters in the first layer to the left of the core has a direct relation to the core, the ones in the second layer to the left has a direct relation to clusters in the first layer to the left and so on. Essentially making sure that services with close relations are positioned closer to each other, and services with longer relations are further away. This follows the same logic as Vizceral, where position is determined based on the importance services has to each other.

![Diagram of data flow](image)

**Fig 3. Illustration of data flow from the left (upstream), to the core, to the right (downstream)**

The size of the circle depends on the amount of microservices in that cluster which makes clusters more easily distinguished. Their size on their own don’t say much, but when combined with position and compared to nearby clusters they can be identified faster the more a user learns the system due to the recognition of patterns, in accordance to Ming-Kuei Hu.

Zooming out the user can get an overview of clusters that have some relation to the core cluster, as suggested by the information seeking mantra. The structure will always look the same for the same core as long as microservice dependencies are not changed or new services are added. By only showing clusters with relations to the core approximately 80-90% of microservices can be discarded.

Since the data is flowing from the left to the right of the “tree”, the clusters have dependencies going the other direction. E.g. the core is depending on the clusters in the first layer to the left because they send data to the core. These dependencies are drawn from a node to another, showing their relations, as suggested by R. Spence. The dependency data is actually what is available and is what the data flow is based upon. When the layout is built, microservice health data is fetched through the Dash service and updates all clusters with appropriate information. A service is deemed unhealthy if at least one alert has gone off for that particular service. Different encodings such as color, size and position is used to give services a direct behavioral look. Nodes that are healthy are colored white and the unhealthy are red, making them stand out and get easily targeted by the user, something called pre-attentive processing. Relations between nodes are represented by lines with different encodings, following classic flow- or tree charts, and is shown in 3 different ways, describing the health. Healthy relations, which is relations where data is sent from a healthy node to another healthy node is shown in white, showcasing ordinary behaviour. Relations where data is sent from a healthy node to an unhealthy node is shown in yellow, showcasing that something might be wrong with the connection, thus giving the user an incentive to look up what might be causing this problem. And lastly relations where data is sent from an unhealthy node to another unhealthy node is shown in red, the most critical status. See figure 4.

![Diagram of relations](image)

**Fig 4. Relations illustration, 1st from left - healthy to healthy, 2nd healthy to unhealthy, 3rd unhealthy to unhealthy**

With the health data incorporated the second automatic filtering process begins. Clusters with only healthy nodes are by default collapsed, effectively hiding them, since they are deemed less interesting from a troubleshooting perspective. A user can click a collapsed cluster to expand it. Healthy relations are hidden for the same reason. That leaves clusters with at least one unhealthy node as well as the yellow and red relations visible.

It’s harder to estimate the status of nodes connected by a yellow relation than white ones since they are not inherently unhealthy. On one hand they might be considered the most important since a yellow relation means that the owner of the node sending data has not been alerted, even though it might be something wrong with it since the receiving node is unhealthy. Conversely, it might also be nothing wrong with the data sending node but only with the receiving one, as a result of circumstances not related to other nodes. Due to this ambiguity, for the prototype, they were decided to be hidden to reduce clutter, leaving only the strongly possible influenced red relations.

Relations where data is sent from a unhealthy node and received by a healthy node is shown as a normal healthy relation since the receiving node is seemingly unaffected.

The application has two main ways of filtering content in accordance to the information seeking mantra. The first is a search function where a user can input an owner or node name, highlighting the results while fading out the rest. The other is four independent buttons that controls the active state of the different relations. As mentioned in the previous section, all but the red relations are hidden by default, but if a user feels the need to see more dependency information the user can toggle the chosen relation. The last of the four buttons toggles relations inside clusters on and off.

To navigate the user can pan around in the tree by holding down the left mouse button and drag, shifting focus between clusters.

### 3.2.1 Design Iterations

The prototype was developed solely with web tools, making it easily available to all developers when launched. Typically modern information visualization is made using libraries such as d3.js [2], which works by creating svg-elements (scalable vector graphic-elements), and is a very effective tool for visualizations with up to around a thousand elements. But when working with
several thousands of nodes, creating several thousand of elements is not very effective. Instead the library p5.js was used. p5.js is a javascript port of the java based processing language, made for efficient visualizations [18]. This made it possible to move the drawing of the visualization to a html canvas (a element used to draw graphics), speeding up the rendering process.

The second prototype was made using a force layout, which works by making different nodes repel each other physically, but are held together using springs as not to force each other out of view. When delving deeper into the development, the realization that a force layout demands very many calculations to position several thousand nodes, even though they are drawn on a canvas, came to mind, and so this idea was abandoned.

The final prototype was built using HTML for markup and showing tooltips, css for styling and p5.js and javascript as the engine. See figure 5.

3.2.2 Prototype Limitations

The prototype relied only on static data due to the reason that real time data would not be necessary to evaluate the proposed solution, additionally it would make it harder to draw conclusions based on user behaviour. By instead utilizing a static, local version of the same data all users was presented with the exact same view, simplifying the interaction observing.

The application was built with the future implementation of real time data in mind. By making use of the exact same data structure as the real time data API, little future development would be needed to incorporate real time data.

3.3 Questions & Task Design

To evaluate whether the prototype successfully answered the research question, a set of questions, ten tasks and a semi-structured interview was formulated. The first set of questions was created to gather background variables, like prior experience and work tasks. The semi-structured interview enabled a retrospective think aloud, making sure the users had expressed their thoughts regarding each task.

Combining the insights gained from the internally conducted interviews (see section 1.2) with the problem derived from the research question, the tasks were designed. They represent typical situations that the developers would have to handle.

By catering the tasks to the expert users, greater support in evaluating if the prototype successfully answered the research question could be made. The tasks can be divided into three categories; localized overview, general overview and relational traversal.

3.3.1 Localized Overview

In the localized overview section, the focus lay on assessing the health of one's own and closely related services. The tasks were designed as to be introductory and could be solved without having to make more than minor configurations. The underlying goal was to establish if the prototype provided observability in the form of overview, i.e., amount of unhealthy services and related owners, as well as if it enabled pre attentive processing. The tasks were formulated as follows;

- Finding and naming one's unhealthy services.
- Finding one's services receiving data from an unhealthy service.
- Finding unhealthy services receiving data from the user.
- Finding a healthy service sending data to a unhealthy service.

![Fig 5. The final prototype. The user is presented with their services in the center. Closely related services are connected via lines to the left and right. Hovering over elements brings up information about them (the names are obscured for this publication). The search field, as well as four buttons to control which lines (relations) to present, is shown at the bottom.](image-url)
3.3.2 General Overview

In the general overview section, the focus lied on assessing the health of owners and services outside the starting view. These tasks demanded more of the user than the former tasks, since a better understanding of the prototype was needed to solve them. The underlying goal was to evaluate how well the visualization provided useful navigational and filtering tools. The tasks were formulated as follows;

- Finding a certain upstream owner and check it’s services health.
- Finding out of the same owner existed somewhere else.
- Finding a certain downstream service and check it’s health.

3.3.3 Relational Traversal

In the relational traversal section, the focus lied on following relations upstream and downstream, establishing affected services as well as the origin of an incident. The underlying goal was to establish whether the prototype successfully answered both research subquestions of providing observability and decision support in the form of dependency insights, and consequently if the insights could efficiently guide the user towards the root cause of an incident. The tasks were formulated as follows;

- Continuation of the former sections last task - Find directly connected services.
- Continuation of the former task - Find indirectly connected services.
- Finding a possible root cause to one of the users unhealthy services.

3.4 User Studies & Analysis

The testing of the prototype was done in two stages. A first small scale pilot user study with the main focus of finding bugs and testing the established tasks. The test was conducted on three people at the Royal Institute of Technology.

The focus for the second, evaluative study relied mostly on assessing how well the prototype answers the research question, rather than trying to find improvements. However, the developers were free to come with any feedback they regarded as appropriate.

When the prototype was deemed ready, the first set of users where picked. The test user base consisted of ten developers from DRMC, all of whom is to be considered expert users since they are familiar with the microservices that constitutes the company’s back-end, and have some form of relation to the already established system of which the prototype developed for this study is aiming to improve upon. They represent different parts of the target group, mainly back-end developers, working in the R&D department. Half of the users are native swedish speakers and the other half speaks english with ages ranging from 25 to 39 years. See table 1. The user studies consisted only of males.

The tasks were executed on a Macbook Pro 15”, using the built in trackpad and keyboard, and the visualization ran on the browser Google Chrome.

According to a study by Jakob Nielsen and Thomas K. Landauer [16], only a somewhat small number of participants are needed in a user study. With five participants you find roughly 80% of all usability problems. With more participants the law of diminishing return will set in and the cost of finding new problems will be drastically higher, since results are likely to repeat. Though, this is mostly relevant to understanding usability issues rather than if the application proves to solve problems.

Each user study was held one on one, with one participant and one moderator. The studies begun by handing out a consent form to the participant by the moderator. The form was modeled from Eric Mao’s “usability test consent form” [31] and Steve Krug’s “usability test script” [32] with minor tweaks to fit this study. The form presents the different parts of the test, establishes that in no way is the user being tested and explains that the test will be voice and screen captured.

When the participant had signed the form they had to answer some initial questions like “what’s your main work tasks?” and “how much experience do you have with Dash?”.

Thereafter they were given a short introduction to the tool by the moderator, explaining how the data flow is visualized, showing how owners and services are clustered, what colors and lines represent, what clicking on nodes, owners and buttons did as well as what the search field did. Subsequently they were given the set of predetermined tasks to be carried out on the prototype.

After the tasks were done a semi-structured interview was held to get a better understanding of their insight into the application. They were asked to share their thoughts when going through select tasks, declaring how they went about solving it as well as providing feedback and critically break it apart. Afterwards three questions was asked with the goal of making the user speak freely about the prototype, not bound to a certain task. They were asked to explain how they felt about the dataflow in the visualization, the cluster placement, as well as if the prototype could help them find root causes to incidents.

<table>
<thead>
<tr>
<th>user</th>
<th>age</th>
<th>work tasks</th>
<th>features or infrastructure</th>
<th>Dash experience (1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>26</td>
<td>back-end</td>
<td>infrastructure</td>
<td>2.5</td>
</tr>
<tr>
<td>#2</td>
<td>26</td>
<td>back-end</td>
<td>features</td>
<td>3</td>
</tr>
<tr>
<td>#3</td>
<td>32</td>
<td>back-end</td>
<td>features</td>
<td>4</td>
</tr>
<tr>
<td>#4</td>
<td>33</td>
<td>prod manager</td>
<td>infrastructure</td>
<td>3</td>
</tr>
<tr>
<td>#5</td>
<td>39</td>
<td>back-end</td>
<td>infrastructure</td>
<td>3.5</td>
</tr>
<tr>
<td>#6</td>
<td>25</td>
<td>back-end</td>
<td>infrastructure</td>
<td>3.5</td>
</tr>
<tr>
<td>#7</td>
<td>33</td>
<td>back-end</td>
<td>features</td>
<td>5</td>
</tr>
<tr>
<td>#8</td>
<td>32</td>
<td>back-end</td>
<td>features</td>
<td>1</td>
</tr>
<tr>
<td>#9</td>
<td>36</td>
<td>prod manager</td>
<td>infrastructure</td>
<td>2.4</td>
</tr>
<tr>
<td>#10</td>
<td>25</td>
<td>full stack</td>
<td>infrastructure</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1. A list presenting the users, their primary work tasks, if they work with features or infrastructure, and their prior experience with Dash.

The participants interaction was recorded by voice and screen capture to later be transcribed. To find common, recurring themes of thought or interaction, the data compiled from the usage of the application was combined with the feedback collected from the users and a matrix consisting of similar statements was created [8]. This could then be used to obtain points of the prototype to improve, remove and redo, as well as constitute a basis for the evaluation of the proposed solution. Three themes; interaction impressions, design impressions, overall impressions as well as 26 statements was developed. The results are found in 4.3 to 4.5.

4. RESULT

This chapter will present the results from the final user study, separated into pre-interview, task observation and completion as well as answers based on the post-interview which is divided into the following themes; interaction-, design- and overall impressions.

4.1 Pre-Interview

Before the user was to execute the different tasks, an initial interview was conducted to collect background variables. See table 1. The information presented in this chapter was needed to validate if assumptions compiled from the previous interviews by DRMC was correct.
In addition to the questions in Table 1, three other questions was asked. To the question “How often are you alerted of an incident when you are on call?” all users answered similarly that it depended on a lot of factors, such as current work tasks, how well the alerts are configured to not get unnecessary alerts, and outside factors like network-, DNS-, or Google issues. The number of alerts ranged from 5 per day to less than 1 per year. The users with the most alerts are working mostly with event delivery and big data, where currently GDPR plays a big role. The user with the least amount of alerts is working mainly with large infrastructure projects, when he gets alerted the problem is probably that a datacenter has gone down, he states.

To the question “What do you do when you are alerted”, the users followed similar paths through the troubleshooting process. Most of the users answered that they usually started by checking the incident summary provided with the notification they got, then check the metrics in Dash, and if they start to understand the problem, they log into into the affected machine to check the accumulated data logs. Some users stated that incidents regarding their services most often depended on external factors, such as DNS-, Google- or general network issues, rather than another service. Those users mostly checked the incidents internal communications channel, trying to collaborate with other people to localise and determine the cause of the incident.

To the question “How long does it usually take to solve an incident?” the users answered similarly to each other, that small incidents take around 5 minutes to 2 hours to solve, while large incidents take around 6-12 hours, and occasionally - weeks. Some of the users explained that most of the time is spent looking for the problem rather than fixing it.

4.2 The Tasks

This chapter will present general observed interactions and user-expressed thoughts regarding the three task categories; localized overview, general overview and relational traversal. See sections 3.3.1 to 3.3.3.

4.2.1 Localized Overview

All users seemed to grasp the concept of healthy and unhealthy services and were able to quickly establish the amount and names of their unhealthy services. One user started by jokingly stating that solving the task would be tedious, demonstrating that he had gotten an overview of the large amount of unhealthy services.

The first set of tasks was finished rather quickly by almost all participants. However, some users seemed to hesitate when trying to determine the direction of upstream and downstream, prolonging the time to finish.

All tasks required the user to bring up certain information, which was an element of irritation, with users stating that “It would be nice to have it [meaning information] directly on the line instead of having to hover[...]

This section’s last task had the user switch between visible relations which seemed to cause a bit of confusion. User #4 stated that “[I’m] not sure if i know the tool to do that.” - user #4.

The users acted most confident in finding their own services, but after a while they seemed confident about finding neighboring services as well.

4.2.2 General Overview

Having to search and pan around to find certain services showed to have a bit of a learning curve. Some users started to manually search for services nearby, rather than using the search field, which prolonged the task completion time. After having spent a while exploring different solutions, they realized that the searchfield could be used. However, the searchfield seemed to cause confusion for the majority of users. User #2 states that “I don’t understand why [owner name] shows up” (translated) and user #4 stated that the owner “[...doesn’t have any services”.

Some users also expressed that they would have liked if the search field showed keywords or auto completed terms.

The majority of users also stated that navigating to the chosen owner by zooming was difficult, and that they “[...would like a click to zoom” (translated) - user #1.

When navigating around in the visualization some users started to comment on the overall health of services, seemingly surprised about the amount of unhealthy services, stating that the cause is probably external.

4.2.3 Relational Traversal

The users seemed to grasp the concept of following lines to find out how services affect each other, with users stating that “[...]the layers are the number of hops” - user #6 and “[...]if we trace all the way, we can find the root[...]” - user #3. However, following relations via multiple services seemed to be the hardest issue to solve. User #1 said that “I chose this node but these other lines are confusing, I expected only to see lines connected to this node” (translated) and “I feel like this is not possible to solve, there are multiple alternatives...” (translated) which was a sentiment shared with the majority of users. Some users got lost when trying to follow lines, stating that:

• It’s hard to relate to what is outside the screen.
• When following a highlighted path, accidentally clicking it away and losing the focus.
• When following relations that seemed to go back and forth.

Some users also expressed concern towards the formulation of certain tasks, explaining that sometimes understanding what was asked of them was harder than solving the actual problem. Because of the stated reasons, the majority of users were not able to finish all tasks.

4.3 Impressions

As stated in section 3.4, the impressions can be categorized into three themes; Interaction impressions, design impressions and overall impressions.

4.3.1 Interaction Impressions

This chapter constitutes feedback and impressions regarding the users interaction, i.e. clicking, hovering, moving the visualization, zooming and searching.

<table>
<thead>
<tr>
<th>Field</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small hitboxes on nodes and lines</td>
<td></td>
</tr>
<tr>
<td>Hovering to get info is annoying</td>
<td></td>
</tr>
<tr>
<td>Unexpected outcome related to search</td>
<td></td>
</tr>
<tr>
<td>Unexpected outcome related to zoom</td>
<td></td>
</tr>
<tr>
<td>Accidentally hid node</td>
<td></td>
</tr>
<tr>
<td>Accidentally hid cluster</td>
<td></td>
</tr>
<tr>
<td>Confusing relations</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. A black box shows users (1 - 10) that expressed a certain statement regarding interaction. Positive statements are colored green and negative are colored red.

The majority of users stated that hovering the cursor over different elements was hard due to their small hitboxes. User #1 and #4 stated that they were not acclimated to using a trackpad
on a MacBook. Four users also mentioned that hovering over elements to show information is sub-optimal.

When searching for owners and services, the visualization zoomed out, showing all clusters but only highlighting the matching results. Though it was possible to specify whether the user was searching for an owner or a service, most ignored that and just inputted the text. This input was handled by the prototype by showing both matching owners and services, confusion ensued when elements were highlighted that did not seem to match. The majority of users tried to click on the area where the highlighted service was, expecting the visualization to zoom in there, causing confusion. By clicking on a cluster it was temporarily hidden, not realizing that, four users assumed that the owner or service was missing data or didn’t exist.

When clicking a node, the visualization focused on it and hid relations not connected in any way. To revert back to the non-focused view, one must press anywhere. Four users accidentally pressed outside the node, removing the focused view. The focused view highlights relations both upstream and downstream, but only on chosen active relations. The highlighted relations might go back and forth in a zigzag pattern which some users found confusing.

4.3.2 Design Impressions
This chapter constitutes feedback and impressions regarding design and UX decisions made during the development of the prototype. Most of the collected information could be categorized into two parts; hindering decisions and blocking decisions. The former is regarded as design decisions that prolonged the time to complete tasks but didn’t completely prevent them from being finished. The latter instead refers to design decisions that made a certain task impossible to finish.

4.3.2.1 Hindering Design Decisions
Four users stated that the initial understanding of which side upstream or downstream was shown could be improved. After spending some time with the prototype, most users seemed confident about their understanding of upstream and downstream.

Some users declared that sometimes it was hard to distinguish between different lines (relations) and between different nodes, that the red color didn’t stand out enough from the dark background. This was especially the concern when the user zoomed out the visualization, mentioning that elements were too small to see.

4.3.2.2 Blocking Design Decisions
Half of the users displayed confusion during tasks when they were to follow relations, searching for the end of the path or neighboring connected services. This lead in multiple occasions to the user not being able to finish the task.

Showing unwanted information was another, related, issue that arose during the user studies, users explained that relations and nodes that were not part of the task or focus were in the way. Again, this lead to some users not being able to finish the tasks.

4.3.3 Overall Impressions
While the feedback on interaction and design decisions mostly came up during the execution of the tasks or the post-interview about them, the overall impressions was gathered afterwards, when the users were to speak a bit more freely about the prototype. The overall impressions is a collection of feedback ranging from different topics, touching both interaction and design decisions.

Table 4. A black box shows users (1 - 10) that expressed a certain statement regarding overall impressions. Positive statements are colored green, negative are colored red and neutral are colored black.

Six users stated that bidirectional dependencies, meaning relations where data is sent back and forth, were not taken into consideration correctly in the visualization. Which in some cases confused the users, prolonging the time to complete a task.

The majority of the users stated that the idea of the visualization was great and something that they truly needed.

Half of the users expressed themselves positively regarding the visualization and all but one user stated that the data flow felt intuitive going in only one direction, though only half the users felt like the cluster placement was intuitive. Four users explicitly stated that clustering services around owners were suboptimal, taking away focus from the important services, and five users stated that their relative positions should be based on the amount of shared relations or data flow between the services.

Three users stated that the visualization is useful as is, helping them troubleshoot and find root causes. The rest of the users said that the visualization could, with a little improvement and polishing, be useful to find root causes.

Six users stated that the application is good for getting an overview picture of the services and relations.

Even though it was stated during the beginning of the interview that a real version of the prototype was meant to display real time data, two users underlined the importance of including real time data to make the visualization truly useful.

5. DISCUSSION
This chapter will discuss the results comparatively to previous interviews and research, establish the preconceived design decisions in regards to the research question as well as propose future solutions to the established problems.

The majority of the users explicitly stated that they found the idea of the visualization to be interesting and valuable to further explore, additionally half of the users expressed positive opinions regarding the visualization approach, stating that “[...]this is gonna be really useful for us.” - user #3, “It’s a very interesting concept and interesting way to visualize it on.” - user #1 (translated) and “I think it’s very valuable to visualize this [...]” (translated) - user #2. However, most of the users indicated that the visualization in its current state is limited, with further development and polishing needed to fully be useful.
Most of the users declared that they found the unidirectional flow of data to be intuitive, though they simultaneously stated that bidirectional dependencies, and in turn the bidirectional data flow, that occurs in the established microservices architecture is not correctly mirrored in the visualization. Only visualizing the flow of data in one direction, information about how services are truly connected could be left out, causing confusion. In the visualization the services are connected via a declared dependency, this in turn suggested that data was sent only the other way, upstream to downstream. As it turns out though, a downstream service first request data from an upstream service that in turn send back the result. When something is wrong with the requested data, that upstream service might send corrupt data back, making the assumption that a service incident always originated from the upstream incorrect. Even if this were to be accounted for, the declared dependencies, or rather lack thereof, might introduce issues considering that users stated that not all services have defined dependencies.

The assumption of a certain type of dataflow is further disproved by the notion from some of the users that not all incidents originated from another service at all. Some users stated that most of their incidents are due to external factors such as network- or Google infrastructure issues, making them untraceable in the current visualization. However some users noted that a lot of seemingly unconnected services were unhealthy, which might be an indication of the influence of an external factor. “If you open the visualization and notice that it’s red everywhere, you can see that it’s a larger issue regarding DNS or Google [infrastructure] [...]” - user #1. Since the impact of external or internal factors differs from developer to developer and team to team, a general tool for troubleshooting both cases needs to incorporate more data than used in this study to be able to correctly show incident origins. The visualization can’t guide a user towards an external incident, but it can perhaps display the occurrence of one. A similar type of overview is what Jon Snow developed during the cholera outbreak and Dahl proposed with his treemap visualization.

The interviews conducted by DRMC before the study concluded that showing owner data was essential to establish who is responsible for an incident, and was the main reason the layout of the visualization was clustered around owners. During the user tests this particular design decision was questioned rather often with four users stating that owner information was unnecessary until the root cause was found. Instead of guiding the user towards the root cause, the owner-centric design brought up a lot of unnecessary data, showing unconnected services and dependency paths seemingly detached from the service which was to be analyzed. This brought a lot of complexity into the visualization which made it hard for users to focus on a certain service and follow it’s dependencies. User #7 stated that “[...] if I look at only one service, I want to see only that service.” Making sure that services, rather than owners, are in focus is what Netflix did with its visualization platform Vizceral, where only direct connected services was shown and upon hovering over the service, additional information was displayed. Removing the owner-centric design will probably streamline the troubleshooting process, making sure the developers find the root cause more often, but only in cases where the incident is internal. With the owner-centric design, the user can more easily get an health overview of the system and perhaps target external problems, as declared in the previous paragraph.

Showing connected services or connected owners both brings up pro’s and con’s. A possible solution is for the developer to decide which view they would rather see depending on their specific use-case. If the developers find that more often than not incidents depend on external factors, perhaps the owner-centric design is most useful, to make sure they get an overview of more services. On the other hand, if root causes often stem from another service, the service-centric design is probably their best bet.

User #1 stated that the service “[servicename] is a library, it can’t break” (translated) which is another factor to take into consideration. The visualization displays nodes without accounting for whether it is a library, a pipeline, a test service not in production or an actual service used heavily, contributing to unnecessary complexity by introducing noise. The clustering of services depend only on the shortest relational path they have, through owners, to the core. If the prototype instead positioned services depending on the amount of data flowing between them, unused test services and the like would be positioned further away, making sure the more important services would be closer to the core and easier to find, which is also how Vizceral determines the distance between nodes.

The visualization has no way of proactively displaying arising issues since the health data is aggregated from already sent out alerts, meaning that developers already know about the incident. This might be an issue since some users stated that sometimes the hardest problems to solve is when a service is misbehaving and no alert has been received, for example when partial queries are failing, which is not accounted for.

The large discrepancy in time taken to determine and solve an incident, as stated by the users, makes it hard to estimate the efficiency of the prototype. With times ranging from 5 minutes to several hours or days, depending on the cause of the incident, a comparison to the time taken to complete a certain task would not yield valuable insights especially since external incidents often prolong the troubleshooting process.

The users seemed to quickly get a sense of the amount of their overall unhealthy services and their relation to neighboring services, showing that some form of pre-attentive processing as well as an overview was enabled.

Further, when users panned around in the visualization they showed excitement and started to explain what different services did as well as commenting on the overall health of the system, showing that the overview reached outside their own services, and that the visualization successfully made the user reflect upon where they were and where they can go.

Going back to definition of observability stated by one user in the previous interviews conducted by the DRMC, that it’s a graph with dependencies and the amount of affected services, it seems like the prototype successfully answered the first sub question; Does the prototype increase observability and decision support?

However, when it comes to the other sub question, Does the prototype help developers find root cause incidents? Generally, it seems like there were too much information shown for the users to successfully navigate multiple layers away from the core. Showing the healthy services made it easy to assess if the amount of unhealthy services was good or bad, but at the cost of losing focus on certain services. Understanding what different relations presented, showed to be somewhat unintuitive, with users answering completely different from each other when it came to finding the root cause more than one or two layers away. Additionally, knowing that some incidents are due to external factors that are not at all represented in the visualization, not all root causes can be found, even with perfect knowledge of the tool.

**Does a information visualization of microservice architecture relations and system monitoring simplify the incident handling process?** The visualization enabled increased observability and decision support but lacked the focus and data needed for the users to correctly determine root cause incidents. The proposed
solution did simplify incident handling to a certain extent, but further research and development is needed for it to be fully useful.

Even though the research conducted in this study is focusing entirely on developers at DRMC, it’s results should be applicable to every business with a large back-end infrastructure. Some businesses, like Netflix, have already established the value of enhancing cognition when determining system health. Visualizations like this is still in it’s initial phase and need of further research to fully explore in and to what extent they are applicable. Microservices architecture relations is not all there is, maybe the relations between nodes can symbolize the logistics of transportational routes between warehouses or the data flowing between nodes can symbolize how much water energy production companies let through it’s dams.

5.1 Method Criticism

This chapter establishes areas in the method that could affect the outcome of the thesis.

The amount of users in the test is rather small, a larger sample would perhaps nuance the insights and bring up new thoughts. The user studies consisted only of males, which might lead to some biased, incorrect assumptions about incident handling and troubleshooting, even though the majority of the R&D department is male. If gender differences play a part in this process is therefore something this study can not tell.

Basing the problem formulation and in turn the development process mostly from the previous interview conducted by the DRMC, lead to some initial design flaws. The addition of another user study at the beginning of the project could perhaps have brought up information which would have streamlined the research and development process by assisting in prioritizing the most important areas.

Some users stated that some tasks were hard to grasp, a initial user study set out to test the tasks on DRMC developers could perhaps have made the studies simpler. By clarifying the tasks, elements of hesitation or misunderstanding could have been removed, probably leading to better results.

5.2 Future Work

With the preconceived design decisions in mind, this chapter will propose future development and research.

5.2.1 Development

There are lots of changes that could be made to improve the platform to make the visualization truly useful. Smaller fixes like changing colors to make elements easier to see, as well making the hitboxes of nodes and relations larger were suggested. Showing service and owner names without having to hover over them would lessen irritation caused by the small hitboxes. Making the search bar more intuitive by focusing on the searched service or owner would also help limit irritation.

Some changes are bigger and necessary to make the platform work in a sensible way. Limiting the amount of services drawn, by either letting the developer choose an overview-view or a focused view. The overview would be similar as proposed in the thesis, the focused view however, will cap the amount of services shown that are not connected to the specific service. This would restrict the amounts of relations shown as well as making sure more information can be visible on the computer screen at a time, lessening the need to pan around and zoom.

Making the application present real time data would be the biggest utility increase.

5.2.2 Research

The addition of a second user study between iterations of development could have given a new dimension of analysis, since the proposed solution could be compared between the studies. Additionally measuring time to complete tasks would give quantitative insights, making comparisons easier.

Suggested future research includes exploring how to render complex datasets while still maintaining focus on certain services and how to account for external factors.

It would also be interesting to conduct an investigative systematic comparison to Vizceral, the tool that is the most similar to the one developed in this study, to better highlight good and bad design decisions, as well as how they compare in their respective fields of visualizing data flow as well as microservice health.

Additionally, it would be interesting to conduct research into how well the proposed solution works compared to other types of information visualizations, in what phase of the troubleshooting process visualizations like this is most relevant and lastly, research into which settings similar types of information visualizations is applicable.

6. CONCLUSION

This was a design oriented experimental research study, set out to explore if an information visualization of microservice architecture relations combined with system health data, could help developers at DRMC find root cause incidents and in term increase observability and decision support.

The subject explored is very valuable and the idea presented with the prototype was positively received with all users, stating that insight into dependencies and the ease of troubleshooting is welcomed. However, some badly preconceived design decisions hindered some of the developers trying to find root cause incidents. While the ability to focus on certain services was lacking, the visualization granted a welcomed overview.

Since the thesis is an experimental research study, there are many advancements to make, such as UX tweaks, better filtering and more advanced interaction techniques. Even though developers couldn’t fully utilize the tool to speed up the troubleshooting process by finding specific root services, the idea and some design decisions had merit which leads to believe a similar but polished tool would make the incident handling simplified.

ACKNOWLEDGEMENTS

I would like to thank my supervisor at KTH, Björn Thuresson for providing me with continuous feedback and hope. My supervisor at DRMC, Jonatan Dahl for helping me establish a subject to explore as well as means of doing so. Thank you to everyone else that helped me with feedback and inspiration, and lastly a big thank you to everyone that participated in the user studies, without you I wouldn’t have anything to present.

REFERENCES


Telea, A.C. 2015. Data Visualization: Principles and