

Visualization of Data Networks

Henrik Persson

DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2020

MASTER THESIS



InfraSight Labs



Abstract In this day and age with so much information available, networks of information, but how do you interpret that information. It is just too much effort to read it all in endless lists, following data relations and analysing it in hopes of fulfilling your use cases. The goal of this thesis is to explore how to more effectively convey information to people through visualizations. It explores different mediums, graphs and other tools that can help with creating a good way to give information to users and help them in their analyses. The thesis describes a work process to get the desired information, and also creation of a functional prototype.

The conclusion is that considering the use cases and the nature of the data is of paramount importance. There were four general use cases which were the ability to see the importance of data nodes, the ability to see what kind of connections they had, the ability to follow the relationships several steps and the ability to see clusters of data forming with your network. It is hard to visualize a network with all the use cases in mind, so users were asked what they prioritized and why it was important. User tests revealed that visualizing the information would help them with some of their work tasks. For some cases it would be enough to have a graph that is shown without interacting with it. But for other cases, adding additional features that enhances the graph in an aspect specifically tailored to the use case solved it for them instead.

Keywords Graph, Visualization, Network, Relation, Data.

Sammanfattning I dagens samhälle finns det så mycket information tillgängligt, hela nätverk av information, men hur tolkar man den informationen. Det är för mycket att läsa all data i oändliga lister, följa datarelationerna och analysera det i hopp om att det uppfyller vad man behöver. Så denna uppsatsen utforskar hur man mer effektivt kan förmedla information genom grafiska visualiseringar. Projektet utforskar olika medium, grafer och andra verktyg som kan hjälpa till med att skapa ett bra sätt att ge information till användare, samt att hjälpa dem i deras analyser. Uppsatsen beskriver en arbetsprocess för hur man kan få informationen man behöver samt skapa en fungerande prototyp.

Slutsatsen som dragits är att användarfall och typen av data, är väldigt viktigt. Det fanns fyra generella användarfall, det var förmågan att se hur viktig en datanod är, förmågan att se vad för kopplingar som finns mellan datanoderna, förmågan att följa kopplingarna i flera steg samt förmågan att se kluster av data som har bildats i nätverket. Det är svårt att visualisera ett nätverk med allt det i åtanke, så användare blev frågade vilka användarfall de prioriterade och varför de tyckte att de var viktiga. Användartesten visade att användarna trodde att en visualisering av deras data hade hjälpt dem i deras arbete. I vissa fall så var det tillräckligt att endast ha en graf som man inte behöver interagera med. Men i andra fall så behövdes interaktiva verktyg som förhöjde grafens egenskaper till att specifikt hjälpa ett visst användarfall.

Contents

Contents	4
1 Introduction	7
1.1 Infrasight Labs	7
1.1.1 Service Mapping	7
1.2 Purpose	10
1.3 Work Process	10
1.4 Related Work	11
2 Background	12
2.1 Usability	12
2.1.1 Aesthetics	12
2.2 Data Visualization	12
2.3 2D vs 3D	14
2.4 Graph Databases	15
2.4.1 Visualizing Graph Databases Use Cases	15
2.5 What Makes a Good Graph	16
2.5.1 Data Relations	16
2.5.2 Scalability/Interactability	17
2.6 PageRank	18
3 Investigation Phase	19
3.1 Basic Functional Requirements	19
3.2 Quality Requirements	21
3.3 Scalability	27
3.4 Graph Analysis	28
4 Existing Products With Network Diagrams	29
4.1 The Different Graphs	29
4.2 Current products analysis	33
4.2.1 Features	34
5 Prototyping Phase	36
5.1 Context	37
5.2 Prototype	37
6 Implementation Phase	40
6.1 Developing Minimum Viable Product	40
6.2 Adding Features	43
6.3 Demo Product Notes	46
6.4 Setting Up the Test	47
6.5 Test Results	48
6.6 Quantifiable Results	48
6.6.1 Feature Requests	50

6.7	Test Result Analysis	50
6.8	Test Discussion	51
6.9	Improvements	52
7	Discussion	56
7.1	Use Case Discussion	56
7.1.1	UC1: Centrality Analysis	56
7.1.2	UC2: Connection Analysis	56
7.1.3	UC3: Path Analysis	57
7.1.4	UC4: Community Analysis	57
7.2	Work Process Discussion	57
8	Conclusion	58
9	References	59
A	Appendix	61
A.1	Test Person A & B	61
A.2	Test Person C	62
A.3	Test Person D	63
A.4	Test Person E	64

Dictionary

- **Node:** A data object.
- **Network:** In this context, a data set of nodes and relations between them.
- **Community:** In data sets there are clusters of nodes that are connected to each other but have few connections outside of this cluster.
- **Service:** A system that supplies a function.
- **Component:** In the context of this report is a data object that could be pretty much anything. A server, service, user, datastore.
- **Relation:** A connection between two data nodes.
- **Degrees of separation:** The amount of relations traversed to get from one node to another.
- **API:** Application programming interface is a communication protocol between different parts of a computer program. In this report it will be how the graph program fetches information from the database.

1 Introduction

There is a theory called six degrees of separation that says that all people in the world are six, or fewer social connections away from each other [1]. So at most someone you know, knows someone, who knows someone, who knows someone, who knows someone, who knows Kevin Bacon. So through whom do you know Kevin Bacon? Well most people do not know that. Because you know who you know, but not everyone someone else knows. Your best bet to get to him would be through the one you know that knows the most amount of people and who does something related to what he does, but who is that? Then there is the Wikipedia game [2]. A game where you are given a random Wikipedia page and through their hyperlinks are to navigate to another random page in as few clicks as possible. In this game, it is still difficult to navigate through the links to your desired destination, and there you get every relation listed. Your best bet would be through links that are in turn connected to a lot of other things, that are also related to where you want to go, but which link is that? This is how a lot of systems work, but to explore relationships this way in bigger data sets is highly ineffective. That is where visualizations come in to play, visualizations can give more easily digestible information to a viewer to aid in their use cases. But what should the visualization show, everything in the network? That would be seven billion people or five million Wikipedia pages. So how much do you show the user? The goal of the thesis is to explore what do you show the user, and how do you show it so that the user can digest the information as easy as possible? What does the user want to know?

1.1 Infrasight Labs

Infrasight Labs is a company that specializes in a single product, vScope, which connects information from complex IT networks to help organizations to visualize important metrics [3]. This product can give their users a report on their components and their services. This however, lacks a good way to explore relations and dependencies between services and components of different types. This can be very important in IT as it can show the root of IT-related problems.

1.1.1 Service Mapping

Infrasight Labs vScope includes a feature, Service Mapping, see figure 1. This feature allows their users to catalogue their services into service cards, (see figure 2), and in those service cards the users can add relationships between their services and their components that vScope has found while inventorying their systems or that is added manually by the user. This gives the user an overview of their systems and it shows a list of all their services and when clicked on it will display everything that service is related to in several lists.

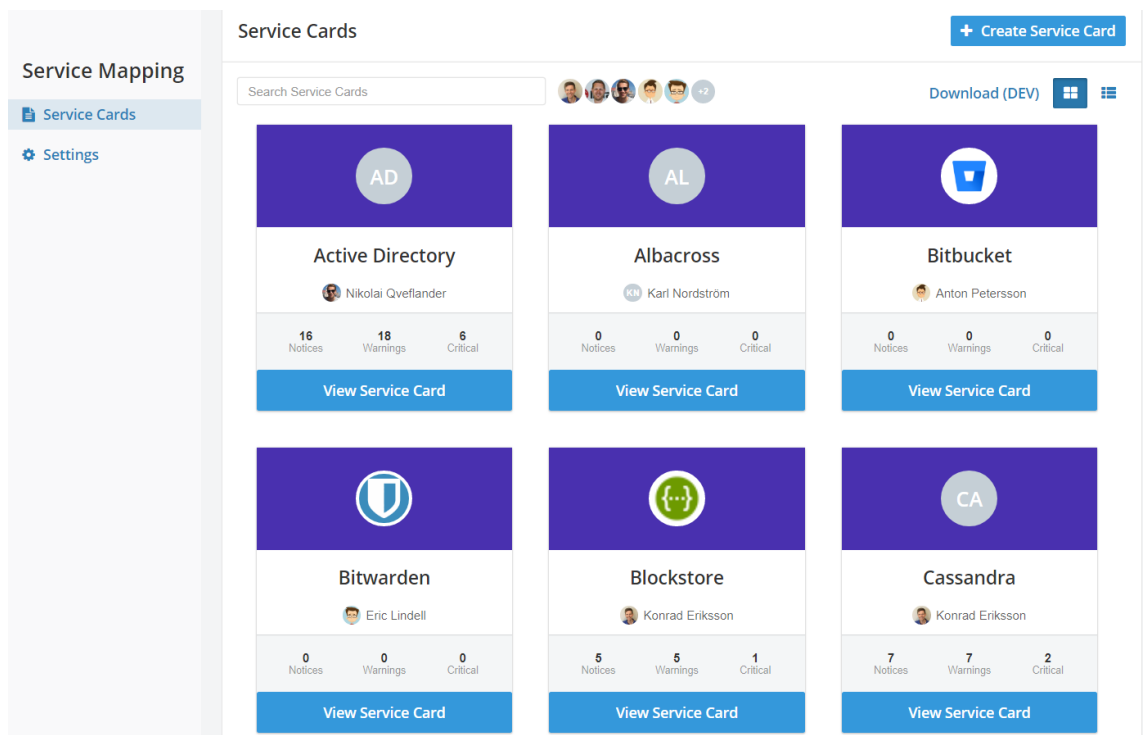



Figure 1: vScope's service mapping, listing service cards





ZooKeeper

Apache ZooKeeper is a service for distributed systems offering a hierarchical key-value store, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed systems.

✕



✎ Edit

About

Owner	 Konrad Eriksson	Environment	Production
Technical Contacts	 Konrad Eriksson	Vendor	Apache ZooKeeper

Instructions for restart Issues "sudo service zookeeper restart" on each VM zookeeper1, zookeeper2 and zookeeper3. One at a time to avoid service disruption.

Service Relationships

Relationship ▲	Description
 Kafka depends on ZooKeeper	-
 Marathon depends on ZooKeeper	-

Servers

Name ▲	OS	Last Patch Date	Comment
zookeeper1	Ubuntu Linux (64-bit)	-	Apache ZooKeeper, node 1. Used by Kafka for keeping track of consumers.
zookeeper2	Ubuntu Linux (64-bit)	-	Apache ZooKeeper, node 2.
zookeeper3	Ubuntu Linux (64-bit)	-	Apache ZooKeeper, node 3.

Figure 2: The service card of an example service, Zookeeper, which shows relationships to other components, and other information.

1.2 Purpose

The purpose of this project is to explore how to create a visualization graph, which will help users to get a better understanding of their networks. Based on the findings, I will create a functional prototype visualizing Service Mapping.

Research Questions:

- How to efficiently visualize a network to a user?
- What to consider when visualizing a network?

1.3 Work Process

This project's design process is divided into three phases, see figure 3.

- Investigation Phase, where existing theory and products will be analysed.
- Prototyping Phase, where the information gathered from the investigation phase is used to design a prototype.
- Implementation Phase, where a graph is made, used to get use cases from the users and tested, then improved.

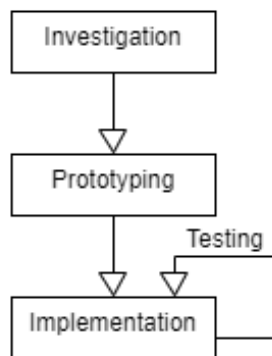


Figure 3: A flowchart of the work process

1.4 Related Work

There are several studies that have been important for this thesis, some of them will be mentioned in this section and their result will be used later in the project. One article explored how to most efficiently show relations and how to structure the visualizations, this article discusses other studies to find out how to do a good graph layout [17]. Another important article researches what users might want from the graphs, here they investigate what users wants to find out from their graph databases [12]. Analysis of design properties, like color and size, as well as how someone can use them is written about in another article [8].

2 Background

Several types of theories are explained in this chapter, which are to be used in the report.

2.1 Usability

In interaction design, usability is of high importance to ensure a good experience for the user and an effective product. Usability can be broken down into several goals [4]:

- Effective to use. Effectiveness is a very general term to describe how good a tool is to accomplish the users goal.
- Efficient to use. Efficiency is a goal to have the user use as little effort as possible to complete their goal.
- Safe to use. Safety refers to keep the user from danger, both in physical form and from making high impact or irreversible errors that might impact data. So both keep the user from making errors but if they still make errors its impact should be low and reversible.
- Have good utility. Utility is a goal where the system should have an appropriate amount of functions to achieve their goals.
- Easy to learn. Learnability is how easy a system is to learn, how long it takes to start core operations and to learn everything the user needs to use to complete their tasks.
- Easy to remember how to use. Memorability is how easy it is to remember how to use the system, which can be different depending on the functions in the system where frequently used functions does not need to be as easy remember as the less frequently used.

2.1.1 Aesthetics

Aesthetics is the visual attractiveness of an object. While usability is about how intuitive functions are, studies has shown that better aesthetics leads to better usability rating from users when all other usability factors are the same. It creates an attractiveness bias which makes the user more forgiving to usability flaws in a product. Users are more inclined to not quit on a product early on if it is visually appealing. However what is and is not aesthetic is more often than not subjective and culturally dependent [5].

2.2 Data Visualization

Data visualization is the graphical representation of data. It is using images to represent data. The images are mostly created for human consumption,

this means that a lot of research in cognitive sciences has been done to determine how a human is most optimized to receive information. Research has concluded that the human eye is good at effortlessly distinguishing differences in position, size, orientation, shape and color [6]. While processing raw tables with just rows of information a graph will remove the reasoning part for the user and do the analysis for them. Using what the human eye is good at both reduces cognitive load and time spent. As can be seen in figure 4 if the biggest number is signified using size, orientation and color it is found instantaneous while the one without any of those indicators take longer to find. Using design principles such as gestalt principles can trick the brain to associate different data points by for example proximity, common region and color [7].

8875318	8875318	8875318	8875318
6581982	6581982	6581982	6581982
2815039	2815039	2815039	2815039
2917750	2917750	2917750	2917750
38120066	38120066	38120066	<i>38120066</i>
11575704	11575704	11575704	11575704
12914651	12914651	12914651	12914651
629120	629120	629120	629120
927030	927030	927030	927030
1885932	1885932	1885932	1885932

Figure 4: Four columns with different ways of highlighting the biggest number, where the first column not having any hints [8].

Visualization properties can be used to represent different things, however the different kinds of properties have varying amount of distinguishable values. While someone can more easily see differences in a lot of different sizes, it might not be as easy to distinguish between several levels of saturation. A table of this can be seen in figure 5, along with recommended ways of using them.

Example	Encoding	Ordered	Useful values	Quantitative	Ordinal	Categorical	Relational
	position, placement	yes	infinite	Good	Good	Good	Good
1, 2, 3; A, B, C	text labels	optional (alphabetical or numbered)	infinite	Good	Good	Good	Good
	length	yes	many	Good	Good		
	size, area	yes	many	Good	Good		
	angle	yes	medium/few	Good	Good		
	pattern density	yes	few	Good	Good		
	weight, boldness	yes	few		Good		
	saturation, brightness	yes	few		Good		
	color	no	few (< 20)			Good	
	shape, icon	no	medium			Good	
	pattern texture	no	medium			Good	
	enclosure, connection	no	infinite			Good	Good
	line pattern	no	few				Good
	line endings	no	few				Good
	line weight	yes	few		Good		



Noah Iliinsky • ComplexDiagrams.com/properties • 2012-06

Figure 5: A table of visualization properties, how many useful values they have, and what they can be used to visualize [8].

2.3 2D vs 3D

Visualizations can be made in both 2D and 3D. Using 3D versions of common 2D data graphs in a 2D environment is usually bad, considering that the perspective might change how a user interprets the data as for example what is closer might appear bigger. If a graph does not add any informational value using the z-axis then it usually is better to create the visualization in 2D instead of 3D [9].

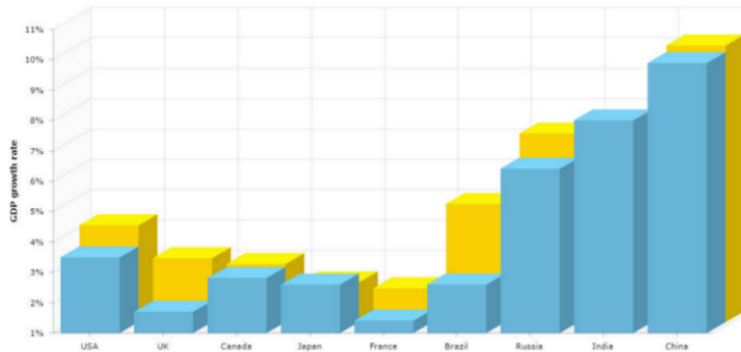


Figure 6: Showing example of a 2D graph made to be 3D [8].

There are advantages and disadvantages with both, one study from Goksör and Kemvik [10] argued that virtual reality could be a great tool to help a user understand relations between large amounts of data. They state however that it might be harder to share the perspective of the user. As in a virtual reality application the user and the user alone can fully see their own perspective. So virtual reality as a tool might be a better alternative if the goal is for one user to understand something, but might be lacking if a user wants to share their understanding or their findings in a data set. Or if several users wants to collaborate to find something.

2.4 Graph Databases

Graph databases are in contrast to relational databases more dynamic as they use nodes to relate data instead of primary and foreign keys [11]. This leads to graph databases being more flexible than the conventional relational databases as they link data between the nodes. Where instead relational databases has lists with common keys that when accessing relations between data needs more advanced queries, especially fetching information that is relations of a relating data node of a relating data node or even more layered data. In that use case a graph database would instead store the relations of every data node [12].

2.4.1 Visualizing Graph Databases Use Cases

There are several use cases (UC) for visualizing a graph database including [12]:

- **UC1 Centrality analysis:** In networks there are nodes that are connected to a lot of other nodes, this can mean that a lot of other nodes are dependent on it so it is the center of a community or network. If for example, a network has a server that most of the services depend on. You may want to see that and decide measures to protect important nodes or distribute the weight.

- **UC2 Connection analysis:** Showing how many nodes are connected to a node and see what kind of relations it has. If it has a lot of dependencies or if it is dependent on others. If for example a famous social media user might take information from a few sources and spreading it to a huge audience or if a user is taking information from a lot of sources but is not sharing to anyone else.
- **UC3 Path analysis:** Showing roads to get to different nodes in connections as well as shortest paths. If for example you have a road network showing roads will help you travel between cities in the shortest path possible.
- **UC4 Community analysis:** In graphs of big data there often exists clusters of information that is connected to each other like communities [13]. If for example you have a big IT-structure you can see patterns of smaller network that act like communities where most of their connections are with each other and not with anyone outside that network.

2.5 What Makes a Good Graph

There are different attributes that can be used to help the user, and all have different features. The positioning of the data object can imply relations and categorizing by placing the data objects in close proximity, it can also be used to order datasets (like in a list) [8]. It has an infinite amount of useful values in the graphs as the data points can be placed anywhere in the graph. The size of the object can imply order and importance of the data point, it has many useful values in the graph. The color can show categorization, while the saturation of the object can display order. Both have relatively few useful values in order to effectively fast differentiate between them. Shape, icon and texture is also good for categorization with more alternatives than coloring. Enclosure and connection using boxes and lines can be used an infinite amount of times and is good at showing relations and categories, where the patterns of the lines and line endings might help with displaying different categories or types of relations. The patterns of the lines and line endings only have a few types of easily differentiable values.

2.5.1 Data Relations

When displaying data relations there are several criterias to ease the perception of graph according to Sugiyama et al [14], Sindre et al [15] and Purchase et al [16]. Some of them listed here [17]:

- Minimize line crossings.
- Minimize line bendings.
- Even spacing between nodes, and lines.

- Even and biggest possible angles between links.
- Place data points that are related to each other in close proximity.
- Maximize symmetry.

Purchase found that minimizing line crossings were the most important of these aesthetics to make it easier for the user to view a graph. However keeping connected nodes straight was even more important for the path analysis use cases as it helps with traceability according to Ware et al [18].

2.5.2 Scalability/Interactability

When the data set becomes too big and the amount of nodes and links increase the graphs become harder to understand. To help mitigate this, a useful tool is interactability. When the user is able to manipulate the graph the user is able to get information at their own pace and only get information that is relevant to them. Features that can be added when adding interactability includes [19]:

- Hover for more information.
- Put data point in center of focus.
- Toggle on/off different aspects.
- Move around and zoom in/out.
- Change complexity.
- Search for objects.

2.6 PageRank

PageRank is an algorithm created by google for their search engine, to not only get relevant pages when the user search for something but also more reputable [20]. So to distinguish the more important pages from the other they implemented an algorithm to set a PageRank value to each site. This algorithm is calculated by giving each site a base value and if a site has links to other sites it increases those sites PageRank by its own PageRank divided by the number of links to other sites it has. So after that, sites that have more links to them are deemed as more relevant. However to make it more accurate it is done again but with its base value updated to the PageRank it was given the last run. This is then done iteratively a number of times, the amount of times depends on the amount of sites, so that relationships between more important sites weigh more. The algorithm also uses a damping factor for the algorithm, but that is basically the algorithm. This is then used when someone searches for something then the user will get a list of relevant results and the pages will be displayed in descending order by their PageRank (more sophisticated algorithms are implemented in googles search results now however). This concept could be used to value node importance of practically any system based on relationships.

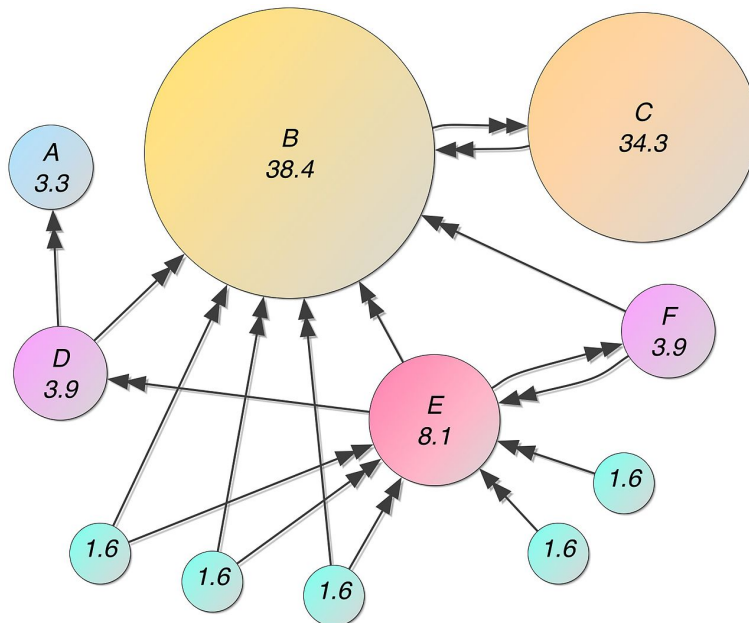


Figure 7: Visualization of a PageRank algorithm [21].

3 Investigation Phase

To start off an investigation begun to see what kind of visualizations would be suitable to solve the project problems. What have been researched before and how does the existing products do their visualization.

3.1 Basic Functional Requirements

Due to the useful features of data visualization it was elected to investigate different kinds of graphs to be used to represent the data to the user. An array of diagrams that could be used to effectively show relations between data points were made (see figure 8).

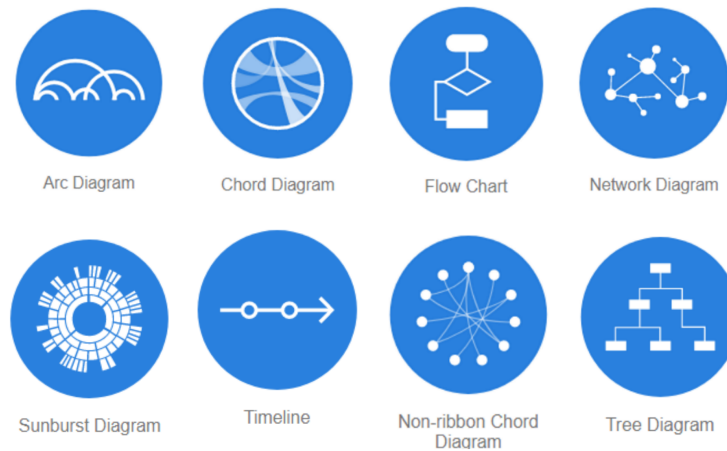


Figure 8: An array of diagrams that can be used to show relations [22].

An analysis of those graphs were made based on its features. Those features had to be relevant to the Service Mapping in the vScope. The most basic features that had to be visualized were discussed with the product owner of vScope. The discussion was a brainstorm session with the product owner, which brought up these four features as the most basic necessities for the graph:

- Allowance for many relationships.
- Supporting several categorizations.
- Show a few levels of relations.
- Show cross relations.

The result from the analysis is visualized in table 1, there it is shown with a check mark if a graph fulfills a basic feature and with a cross if it does not.

Table 1: Various diagram types, and functional requirements.

Diagram Requirements				
Diagram type	Many relations	Several categories	Few levels of relations	Cross relations
Arc diagram	✓	✓	✓	✓
Chord diagram	✓	✓	✓	✓
Flow chart	✓	✓	✓	✗
Network diagram	✓	✓	✓	✓
Sunburst diagram	✓	✓	✓	✗
Timeline	✗	✓	✓	✗
Non ribbon-chord diagram	✓	✓	✓	✓
Tree diagram	✓	✓	✓	✗

The graphs that covers all requirements are arc, chord, network and non ribbon-chord diagram (see figure 9).

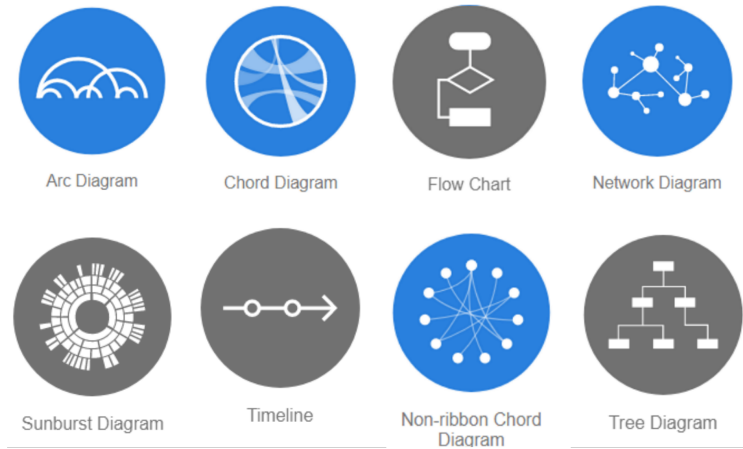


Figure 9: An array of diagrams that can be used to show relations [22].

Those diagrams can show many relationships with its lines, use coloring and placement to display categories and the lines can use different color, line

patterns and line endings to show different kinds of relationships. Every data point also shows its relation to other data points for cross relations.

3.2 Quality Requirements

With the basic functional requirements out of the way we can see what kinds of graphs can do the job. However to see which one is the best, there was an investigation using quality requirements stemming from previous research in the field.

Table 2: Diagram types and quality requirements.

Diagram type	Diagram Requirements						
	Minimize line crossings	Minimize line bendings	Even spacing between nodes	Even space between lines	Even and as big angels as possible	Place data in proximity	Symmetry maximization
Arc diagram	✗	✗	✓	✗	✗	✓	✗
Chord diagram	✗	✗	✓	✗	✗	✓	✗
Network diagram	✗	✓	✓	✗	✗	✓	✗
Non ribbon-chord diagram	✗	✓	✓	✗	✗	✓	✗

With the table 2 an analysis was made to see which of the previously stated quality requirements of the data relations were able to be fulfilled with the diagrams that met the functional requirements. The data relations quality requirements were:

- Minimize line crossings.
- Minimize line bendings.
- Even spacing between nodes, and lines.
- Even and biggest possible angles between links.
- Place data points that are related to each other in close proximity.
- Maximize symmetry.

The table was made by creating example graphs that visualized a special scenario. This scenario visualize an overly simplified case that could occur in Service Mapping, and it is just an example to illustrate how well the different graphs could handle such a case. The scenario was: A service (Service 1) has three servers (Server 1, Server 2 and Server 3), three different users (User 1,

User 2, User 3), two other services (Service 2 and Service 3) that has two of the users (User 1 and User 2) that the other service has and two of the servers (Server 1 and Server 2). This a very small scale scenario of what could be needed to visualize in the finished product. And the results were these graphs of the arc diagram (Figure 10), chord (Figure 11) and non-ribbon chord diagram (also doubles as a network diagram as it would look the same in this scenario) (Figure 12).

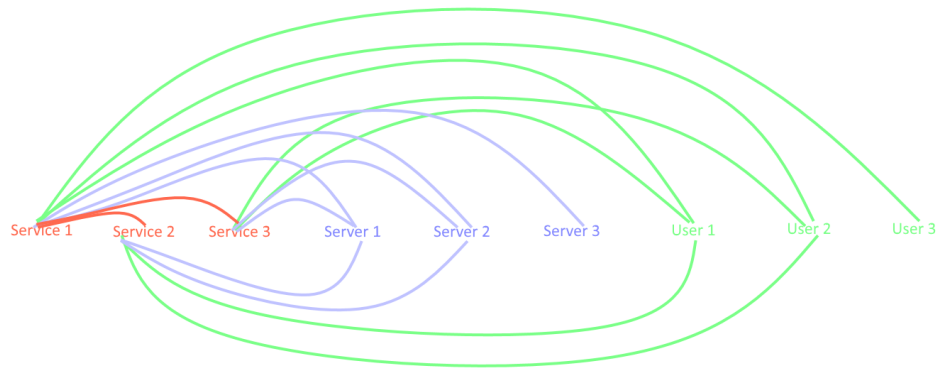


Figure 10: Graph of an arc diagram visualizing a set scenario.

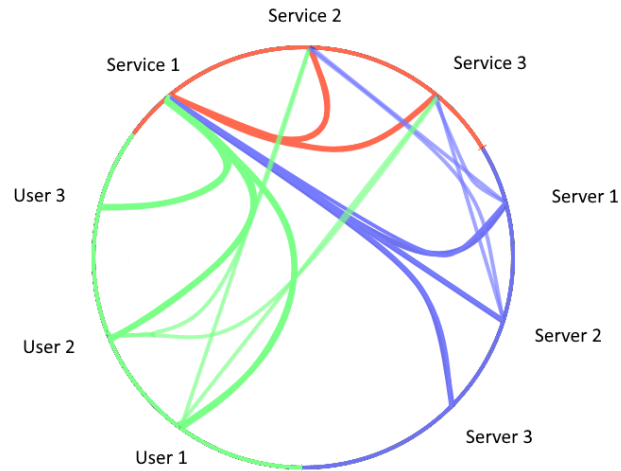


Figure 11: Graph of a chord diagram visualizing a set scenario.

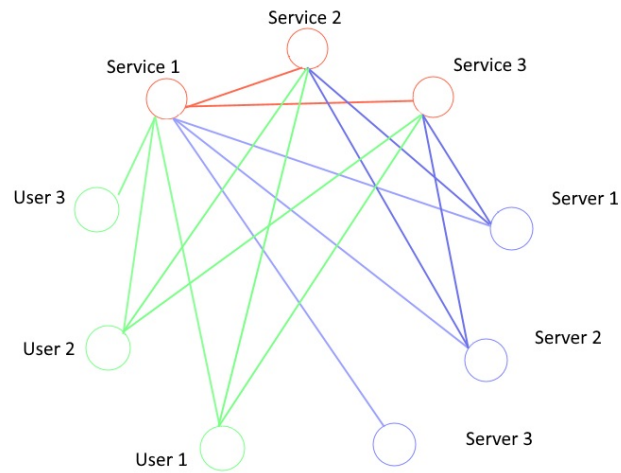


Figure 12: Graph of a non-ribbon chord- or network diagram visualizing a set scenario.

Those graphs did not fulfill many of the quality requirements, neither graph passed even half of the requirements as the data created too many relations.

Neither graph was able to have few line crossings as the cross relations were too many even in this small scenario case. The line bending case was just relevant for arc and chord diagrams, as they inherently depends on having bent lines in their visualizations while network and non ribbon chord diagram does not. Even spacing between nodes is easily managed by all graph types in this scenario. However the space between the lines and their angles is completely decided by where the connecting nodes are and cannot be guaranteed by any graphs. All graphs were successfull in placing the categorically relating nodes grouped up to signify that they are of similar type. But it was not possible to make the graphs appear symmetrical in its connections due to the chaos of all connections relating to each other everywhere.

With the failure of the first graphs another set of graphs were made to display graphs where they only visualized the relations of Service 1 in full (one degree of separation) and the other services relations having lower opacity to help the user by making them only focus on one thing at a time while still letting them explore other paths if they want. This would require intractability to function as the user would need to change which node to highlight. The results were this arc diagram (Figure 13), chord diagram (Figure 14) and non-ribbon chord/network diagram (Figure 15).

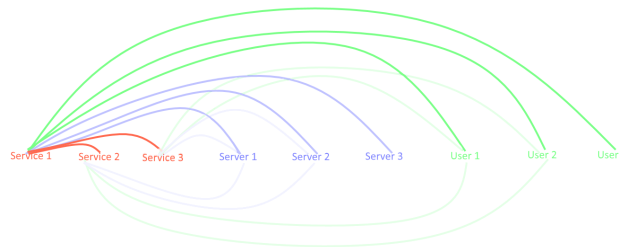


Figure 13: Graph of a arc diagram visualizing a set scenario.

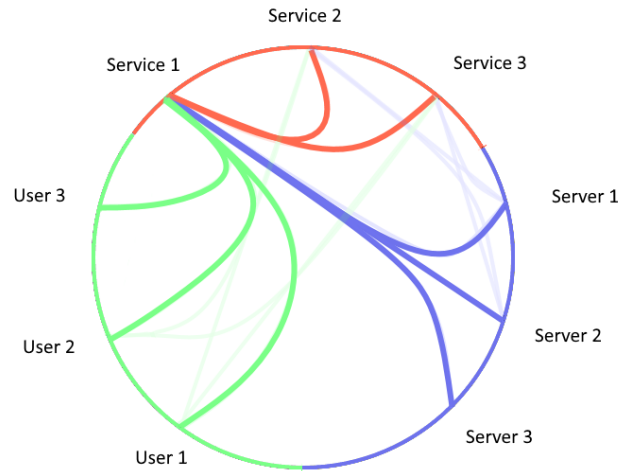


Figure 14: Graph of a chord diagram visualizing a set scenario.

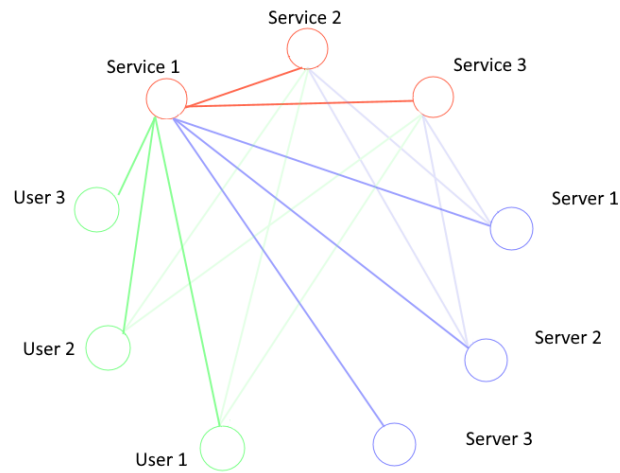


Figure 15: Graph of a non-ribbon chord- or network diagram visualizing a set scenario.

Non-ribbon chord diagram and network diagram would look pretty much the same and function the same with these requirements, however if a certain

node is the object of focus another kind of network graph can be made like (Figure 16) where you could show every relation in clusters to their respective categories.

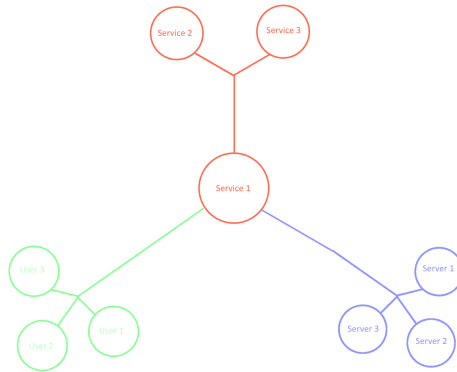


Figure 16: Graph of a network diagram visualizing a set scenario.

This would require interactivity to scale up to a bigger network and to display relations of relating nodes. One way to do this scaling is to have a preview of the nodes relations on for example hover like in Figure 17.

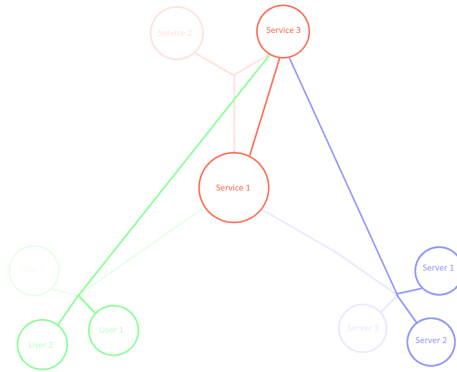


Figure 17: Graph of a network diagram visualizing a set scenario.

This would allow for the network to be consistent but it would not guarantee the quality requirements of angles that are even and as big as possible. Whereas in the figure 18 it is made to show the hovered nodes related nodes in the same way as the original model.

as it is with all its countries (communities), but the user can also zoom in to a country (community) and see the roads (relations) between cities (nodes). The same could be done with the other diagrams, this would then zoom in on a section of the graph and have all nodes in a community placed close to each other. However the community relations between each other would be harder to see.

3.4 Graph Analysis

When the data becomes bigger, network diagram becomes the only practical way to visualize networks mostly due to its abilities to scale. But for a network with a small data set one might want to prioritize aesthetics and go for a chord diagram which arguably look prettier. In this case one might be able to bend the quality requirement rules in favor of aesthetics as it would still be possible to have the graph legible anyway. But this report will continue with network diagrams because it is more practical data sets of varying sizes.

4 Existing Products With Network Diagrams

First to note is that most of these graphs are made for developers to create their own features and visualizations and the analysis of the graphs is based on visualizations that may or may not be supported by the companies that makes the engines. But all these graphs are taken from websites related to the company stated. Also to note is that the data that they visualize is not the same in either case as it would bring too much work to create a database that is readable by all the tools. So the examples all have their own sample data which may or may not make the visualization look better than it is. Also to note is that this is not a comparison of these tools to see which one is better, rather only to find out what features existing visualizations have and to find out what kind of layout and features to include to best fulfil the usability goals and the ability to complete the use cases of visualizing graph databases.

4.1 The Different Graphs

There are several products that specializes in visualizing graph databases. Keylines is one product by Cambridge Intelligence [23]. They usually use network diagrams with interactability. There the user can combine nodes and create filters and add scripts to suit the specific data set that the user works with. An example is figure 19 .



Figure 19: A graph of a network diagram made using Keylines

Another product is ConnectTheDots by Databasics [24] which also uses network diagram that has a table beside the graph to show a list of connection and its centrality number that means that if a user wants to travel in the diagram these dots are good to use. And it shows how many connection each node have. Clicking or hovering over a node will show that nodes connections and central-

ity number and lowers the opacity on every node that is not directly connected to the highlighted node. Example in figure 20.

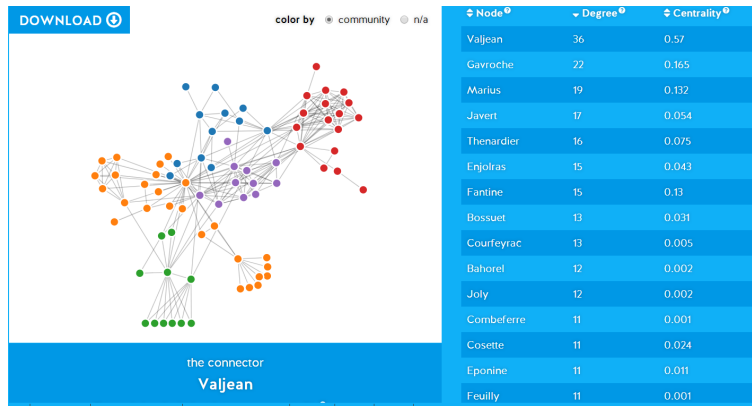


Figure 20: A graph of a network diagram made using ConnectTheDots.

Alchemy.js by GraphAlchemist [25] also uses network visualization with the interactability to show drag around nodes and the having its related nodes dragged along with it to customize the view. It also has the ability to add features using Javascript. Example in figure 21.

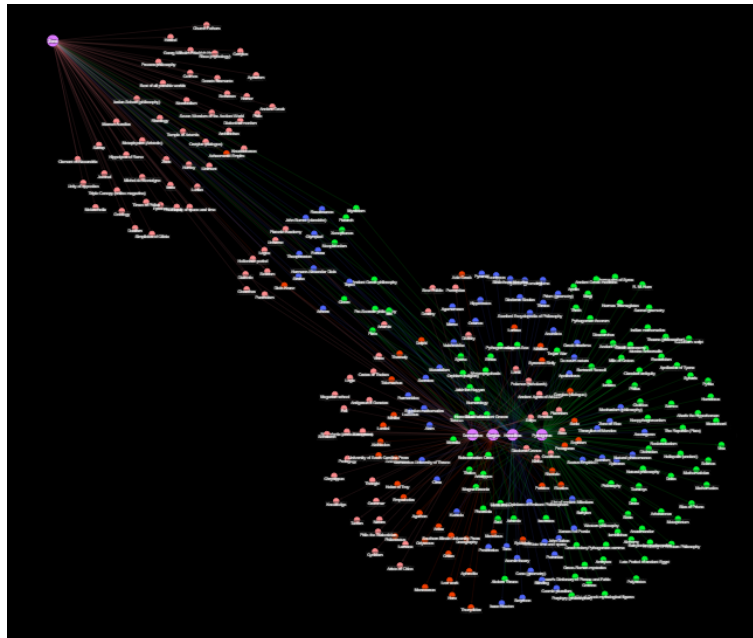


Figure 21: A graph of a network diagram made using Alchemy.js.

Another example is Kumu by Kumu Inc [26] which for the most part is used to create network diagrams. In this graph the network, which is very simple consisting of very few relations between objects, uses containers to group categorically similar nodes, different kinds of lines to signify different kinds of relations. It has a legend describing the different lines and a table that lists the information on the left part of the screen if a node is selected. It also gives the option to only show one, two or three degrees of relations from the selected nodes as well. Example in figure 22.

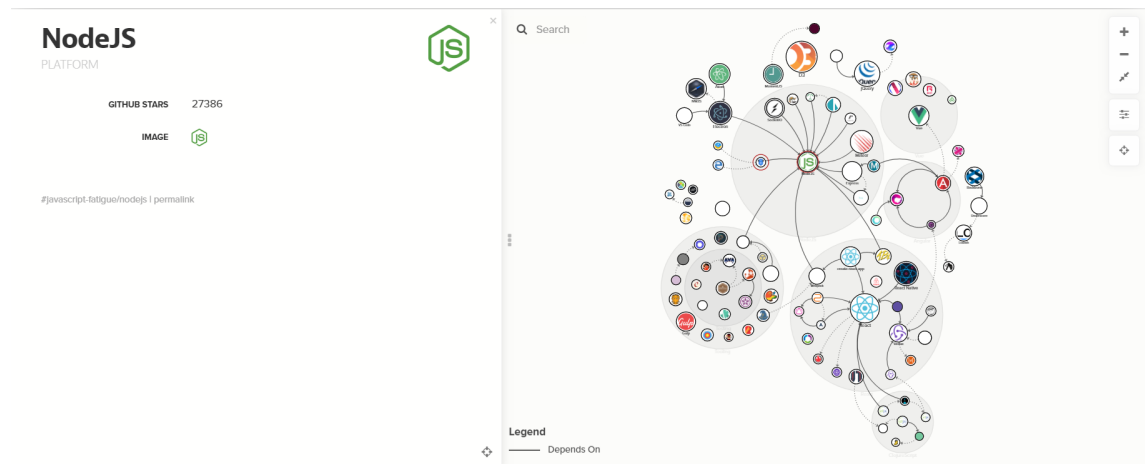


Figure 22: A graph of a network diagram made using Kumu.

Infrasight Labs has their own graph visualizer internally using a network diagram, this visualizer uses icons to categorize. Visualizes data in communities. It has features to show data in different times, ability to filter out certain kinds of nodes and categories. It shows information on hover and ability to drag around nodes. Dragging around nodes does not affect any other nodes positions like alchemy.js.

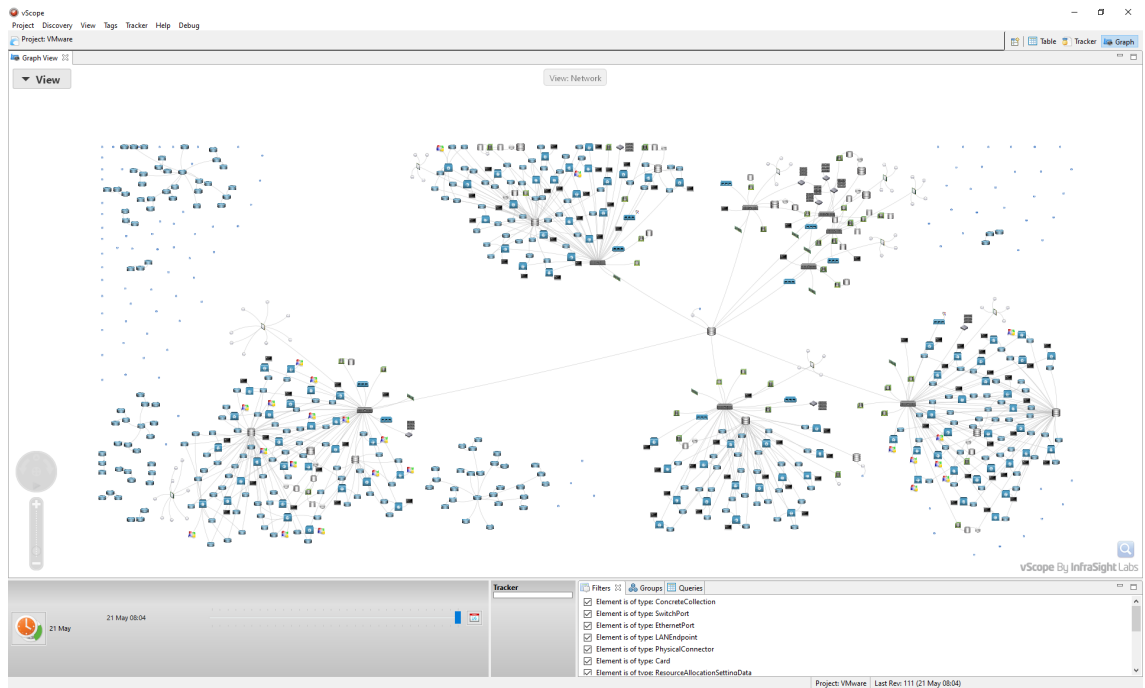


Figure 23: A graph of a network diagram made using inhouse software at InfraSight Labs.

4.2 Current products analysis

The products discussed all have differences in features. They have been analyzed like the other diagrams to see if they fulfill the different quality requirements. The results are shown in table 3.

Table 3: Comparison of existing programs and whether they meet the quality requirements.

Diagram Requirements							
Diagram type	Minimize line crossings	Minimize line bendings	Even spacing between nodes	Even space between lines	Even and as big angels as possible	Place data in proximity	Symmetry maximization
Keyline	✗	✓	✗	✗	✗	✗	✗
ConnectTheDots	✗	✓	✗	✗	✗	✓	✗
Alchemy.Js	✗	✓	✗	✗	✗	✗	✗
Kumu	✓	✗	✓	✓	✓	✓	✗
Infrasight Labs	✓	✗	✗	✗	✓	✓	✗

One thing to note is that while the Kumu graph does fulfill a lot of the requirements, some of them would be very hard to fulfill it if the data were to be more complex with more relations between objects. The data that the graphs has been tested on has not been consistent as it has been their own test data in each case. Which gives the graphs different levels of difficulty to complete the quality cases. Whereas Kumu's graph has good checks it has very low complexity in the sense that there are quite few data nodes and very few relations. So for this part it is very good at displaying the test data set but with more relations and nodes it would probably clutter up too much (that said, this is what graph visualization is all about, understanding the data set and show it accordingly). Both Kumu's graph and Infrasight Labs graph does a good job at aiding community analysis. Infrasight does it by placement, each cluster is a community whereas Kumu uses the grey boundaries to indicate a community. ConnectTheDots and Alchemy.js uses color coding for their communities which also makes it clear that the nodes are connected in some way. Keyline, ConnectTheDots and Alchemy.js has very messy line crossings everywhere which makes the relations harder to do path analysis on, Infrasight Labs also has this problem at several places however it is okay at others. Regarding Centrality analysis Kumu does a decent job by having their central nodes in the middle of the communities. Alchemy.js uses different sizes to show the most central nodes which is a good way to get the attention of the user. ConnectTheDots does not use visual elements to help the user in this task but instead uses the table to show centrality, which makes it easy to find them but it does not give

any visual clues so it is not viewable from the graph and instead the user clicks in the sidebar to highlight the different nodes to see their centrality. Using the table to help with the centrality analysis however is a bit worse in a usability standpoint as it does not give visual overview in the graph, instead the user has to cross check with table to find the centrality. As for Infrsight Labs there is no clear design property to support it however it has visual cues that where the lines are thicker there is a lot of connections as there are lines that are close to each other, this however would not be a desirable trait otherwise as the clumped nodes makes path analysis harder. Regarding Connection analysis only the Kumu graph has any indication on what type of connections the nodes are having and only Kumu and Alchemy.js has any indication of the direction of the relation.

4.2.1 Features

In addition to the quality features there is room for extra features that can help the user to get a better view of the graph. An analysis was made for each of the five products to get an understanding of how they worked as well as their features and layout. Here is an analysis of what tools and functions the graphs are using and a personal conclusion was made of which quality and functional features that were intuitive and understandable. This entails that the analysis was not the same for each product, instead based on their characteristics and what the product should be used for. This was done to give an idea of how tools and features can be used to fulfill the use cases.

Some aspects that were observed during the analysis was that Keyline uses size on lines and objects to signify importance in relations and nodes. Connect-TheDots categorizes by color and has a table on the right hand side to select certain objects, when selected it fades the other nodes to highlight the selected node. The table can be sorted in different ways to find nodes of interest, where default is to list the most central nodes. The Alchemy.js example has the feature to drag around nodes to place them as the user wants. When dragging around nodes the nodes connected to it follows. The following nodes gets closer to the followed node depending on how many other nodes it is connected to. The Kumu graph has a table on the side that displays information of a highlighted node that is selected either by clicking on a node or using the search bar. It has a legend describing the different lines. It also gives the option to only show one, two or three degrees of relations from the selected nodes. Infrsights graph visualizer has the feature to filter out categories of nodes using its sidebar and has the ability to show data at different times. It shows type by icons and displays nodes in community clusters.

According to this analysis Kumu does a great job with all the stated common use cases for graph database visualization. It even arguably has the best aesthetics of all the analysed graphs. However Kumu's data set is very small and would probably not work as well with a bigger data set with more relations and so on. While it does not follow the quality requirement of minimum line bendings its curved lines might actually benefit it in usability through increased

aesthetics as the bent lines follows the through and through theme of roundness. But it might make the path analysis harder. Infrainsight labs graph has a more robust data set and does a good job at showing communities, however it has a very hard time with path analysis as the lines are bent, have the same shape, pattern and color and they are very cluttered at places. Keylines does path analysis better as the lines have different sizes which help the user trace it cluttered places. As it would seem that in graphs with big data sets it seems incredibly difficult to draw a whole network without crossing lines, so to mitigate the damage some sort of way to distinguish the different lines would probably help. One thing that Alchemy.js, Kumu and Keyline has is text labels under each nodes to help the user identify a node without hovering it.

5 Prototyping Phase

With the information gathered from the investigation phase examples, certain features were suggested to help with the different use cases. As a network diagram was decided to be the best choice to move forward with. Since it was the best graph with the quality requirements and scalability. Several features were listed to help the different use cases. Each feature followed by how many different usable values each feature has. From many, medium and few [8].

- Path Analysis
 - Straight lines, many.
 - Line color (To differentiate close lines), few.
 - Line patterns (To differentiate lines close to each other), few.
 - Line thickness (To differentiate lines close to each other), few.
 - Position (Spacing), few.
- Connection Analysis
 - Arrows (To see direction and what kind of relationship), few.
 - Line color (What kind of relationship), few.
 - Saturation/fading (To see direction), few.
 - Line pattern (What kind of relationship), few.
 - Line thickness (show importance or strength of relationship), few.
 - Size (Size might mean how many nodes rely on this) , many.
- Community Analysis
 - Position (Community by placing nodes in clusters), many.
 - Color (Same color for nodes in a community), few.
 - Background (Have a background color in clusters that boxes in the nodes), many.
 - Shape/Icon (Same shape/icon for nodes in a community), medium.
- Centrality Analysis
 - Size (Size might mean how many nodes is connected to the node), many.
 - Position (Put it in the center of clusters), few.
 - Angles/spacing of lines (So the user may focus on clumps of lines to see central lines) , few.

There are more things that a feature can be used for example. To:

- Identify a node

- Text (Displaying the name), many.
- Shape/Icon (Showing the category or icon of node), medium.

Some of the features are distinct to one type of use case while others are a used in more than one. Color, size and position is the most listed features for different use cases. And while position can still be used in all use cases without interfering with each other (nodes, can be spaced out, placed in communities and central nodes can be centered in clusters without confusion). Color may not have that ability. As if you give colors different meanings, for instance if the color blue means that it is a part of a community, but the color blue on a line means a special sort of relationship it might cause confusion for the user as the same color might make them think it refers to the same thing. Size should mean the same thing too, so for node size you could either set the size based on the amount of connected nodes, including both dependencies on other nodes and how many nodes depend on the node, or just how many nodes depends on it. Different line patterns, colors and thickness could benefit both path analysis and connection analysis. However for the path analysis it just helps that they are different. If they are different to help connection analysis it would help path analysis too in the case that all are not the same type of relation. You could use both shapes and icons for nodes so there is room for two versions of that if you need to.

5.1 Context

The diagram that is drawn should have different features depending on the nature of the data and the prioritization of use cases. As seen in the Kumu example it is very fitting on small amounts of data and relationships. And Infrasight has a better graph with bigger data sets and they use very different design approaches. A designer should therefore consider what features they use at different places given the importance of the different use cases and the amount of and different categories of data and relations. If a data set only have one kind of relation or it is non-directional then it might be recommended to only use one color for the lines and save the coloring for community or categorization. And if the only important thing for the user to see is one node and its closest relations then it is not necessary to clutter the graph with all other nodes. If the use cases are more leaning towards either connection or centrality analysis, then size could be used to indicate the more important one. So to design the best possible graph visualization one should listen to the end users use cases and have an understanding about the amount of and the nature of the data.

5.2 Prototype

Using the context information, a low-fi prototype was created, see figure 24.

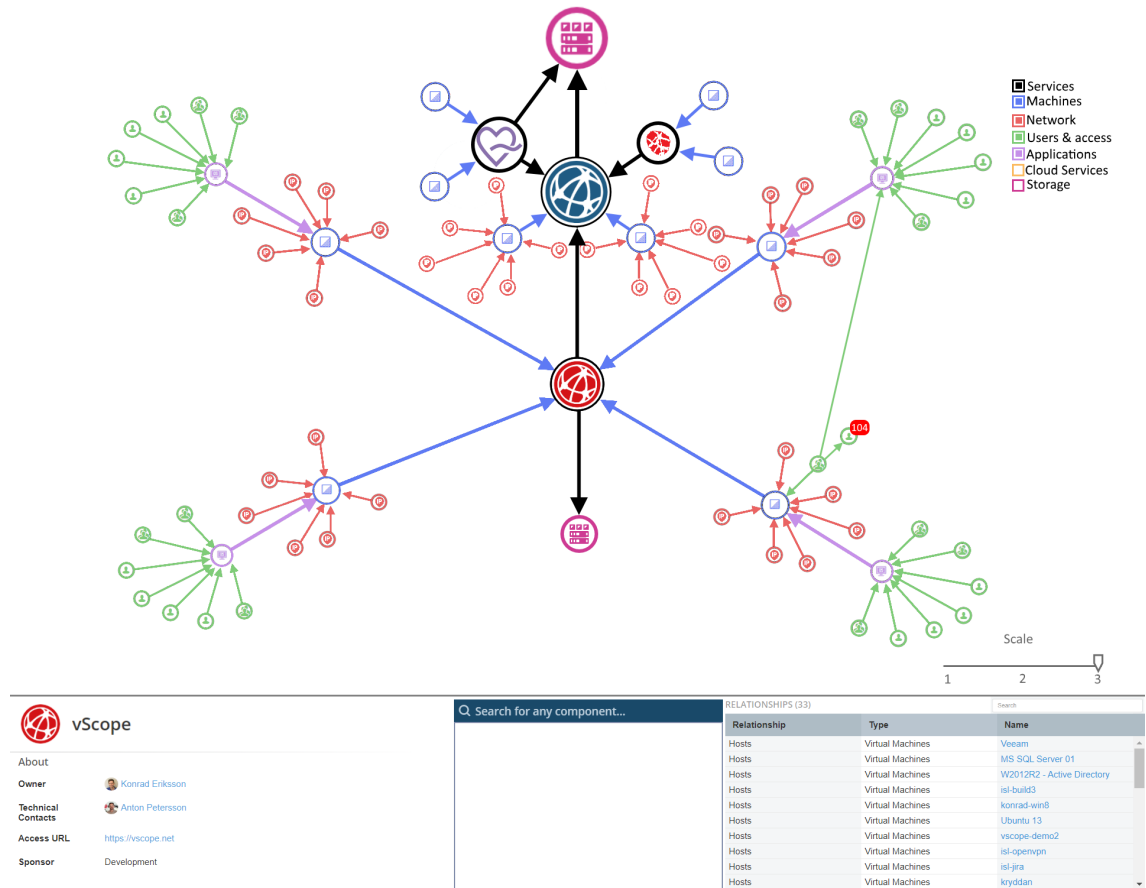


Figure 24: Low-fi Prototype

Here it was prioritised that the node in focus should be centered so that the user knows what view it is. Use the size to indicate importance of the nodes. In Infracore Labs own system, vScope, they have different colors and icons for all different kinds of components like in figure 25, except for their services.

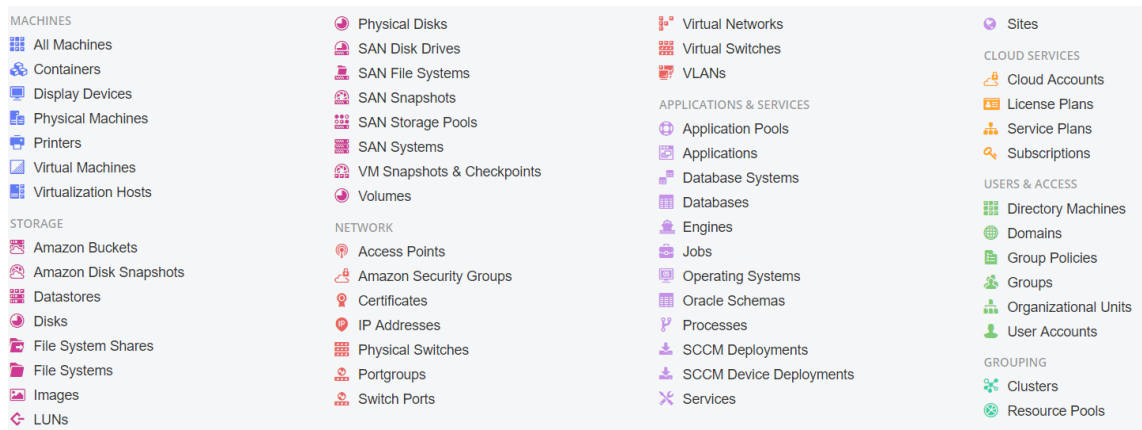


Figure 25: Icons of all components.

This was used to display the different kinds of nodes as it was assumed that the users would recognize the icons and have some idea of what they represented. As services did not have any specific color or icon it was made black. The relationships are shown with lines and arrows to the target node and is in the color of the node that has a dependency. What is shown in it is servers, IP-addresses, services, datastores, applications and users. A legend where the different items could be toggled on and off. And various tables of information regarding the selected node is in the bottom of the graph. It also has a slider that selects how many degrees of separation from the selected node that will be shown.

This prototype was tested on employees on Infrsight Labs, which they thought was an adequate solution. However they thought that to give the users better perspective something more tangible and interactable had to be developed. Both to help their understanding and so they got a peak of the feature that they could later use, so that they got something out of the meeting as well.

6 Implementation Phase

As context is so important to know when developing the visualizations there was a need for the users input. When users heard of the service mapping feature which this program is about visualize, several of them expressed a need for a graph that could visualize the dependencies without knowing it was in the works. However when asked about what they wanted it for they did not know, but they wanted it. Answers were mostly in the sense of 'It would be nice to see how things are connected' or something similarly vague. So to get the most of the user input a different approach was taken where a working prototype was made to give the user something to draw their ideas from. As what was most important to the project was to see what their most wanted use cases would be. As there were limited amount of contacts and time with them we thought that the best course of action would be to get the most amount of information from what the user wants and as well get some sort of usability test, was to display a kind of minimum viable product (MVP).

6.1 Developing Minimum Viable Product

A development started of a MVP. It was developed using the same tools that Infrasight Labs use to ensure compatibility with the rest of their product. The product used their API to display nodes representing components in their system. It took a service component and visualized everything that it is connected to. However this was not practical as the amount of data cluttered the screen and from just one degree of separation the whole screen was full and a lot of the objects were outside the screen. So as a start before the user has expressed which type of components they want to see it was limited to just services, servers and datastores. The servers and datastores got their icons and colors from the table in figure 25 while services were black. A lot of services did not have dedicated icons, but if there already were an icon representing it, that would be used, otherwise it would be its two signature letters (first letter in the first two words or the first two letters in their only word, depending on the name of the service. This is how it is represented in their Service Mapping tool). One service is chosen as a starting point and then the program fetches its related services, datastores and servers. It then continues to fetch its related services datastores and servers related components iteratively until the "whole network" is shown (it wont be the whole network as everything might not be related to each other and every relation is not explored). The nodes were then plotted on the screen with even angles and space between them from the starting node and then for every layer added it would share a set angle to have its nodes easier to follow, as to fulfill the quality requirements of even angles between lines, even space between the nodes and the ability to have as straight lines as possible so it is easier to follow the relationship path between several nodes. This worked really good in some cases, but in cases where there were too many nodes with relations to a single node and it tried to have as straight lines as possible it would clump up and nodes would end up inside each other. And in other cases nodes would

clump up inside each other if there were two nodes that would be close to each other and both have a lot of relating nodes, as shown in figure 26.

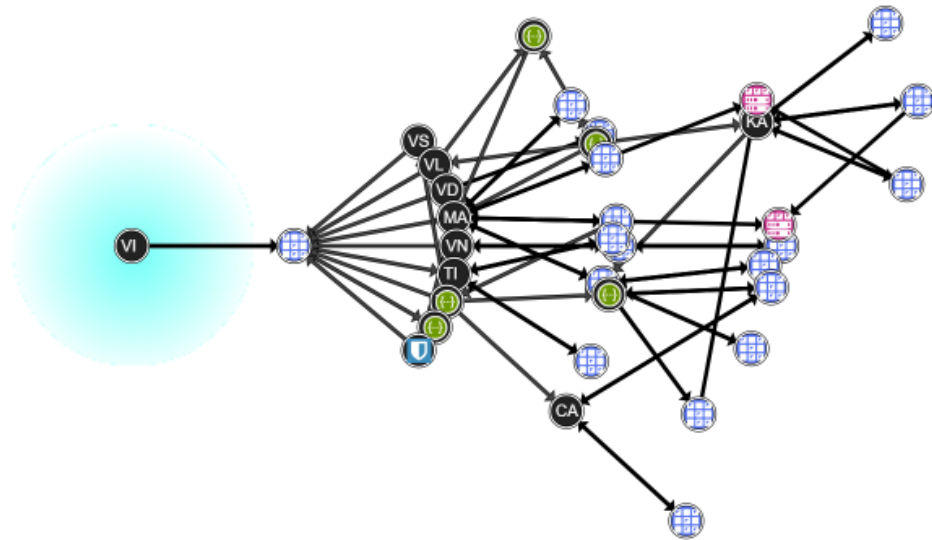


Figure 26: Graph where the design makes nodes appear inside each other.

So to circumvent this a force was added to the nodes so that the nodes were repelling each other, this however made the angles of the lines more uneven less controlled, however it was very necessary to not have nodes inside other nodes so it had to be done. However it also had a quality requirement benefit as it would place nodes with more even spacing between them, see figure 27.

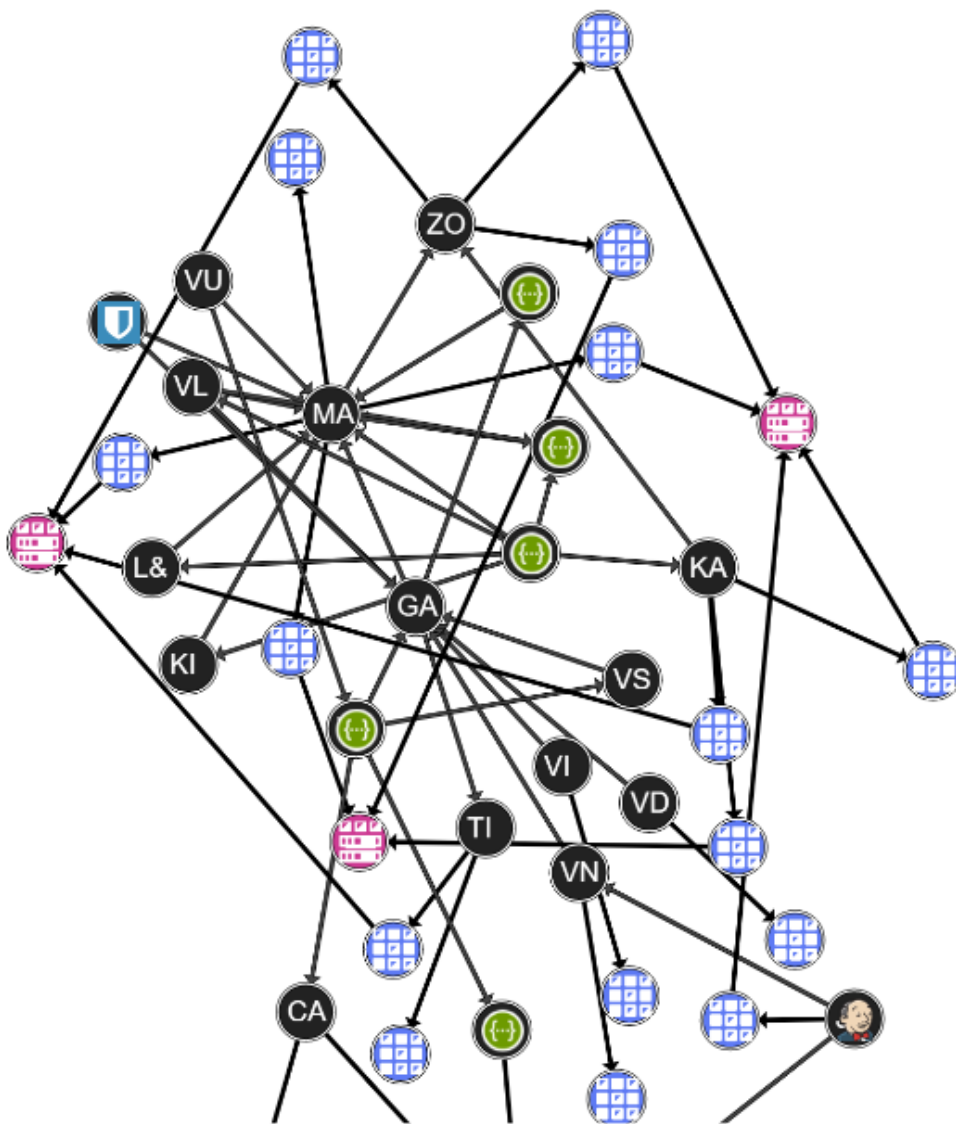


Figure 27: Graph where forces are added to prevent nodes to appear inside each other.

A slider was also added so that it was possible to fade in different levels of relationships in the graph. By levels meaning degrees of separation from the original node, see figure 28.

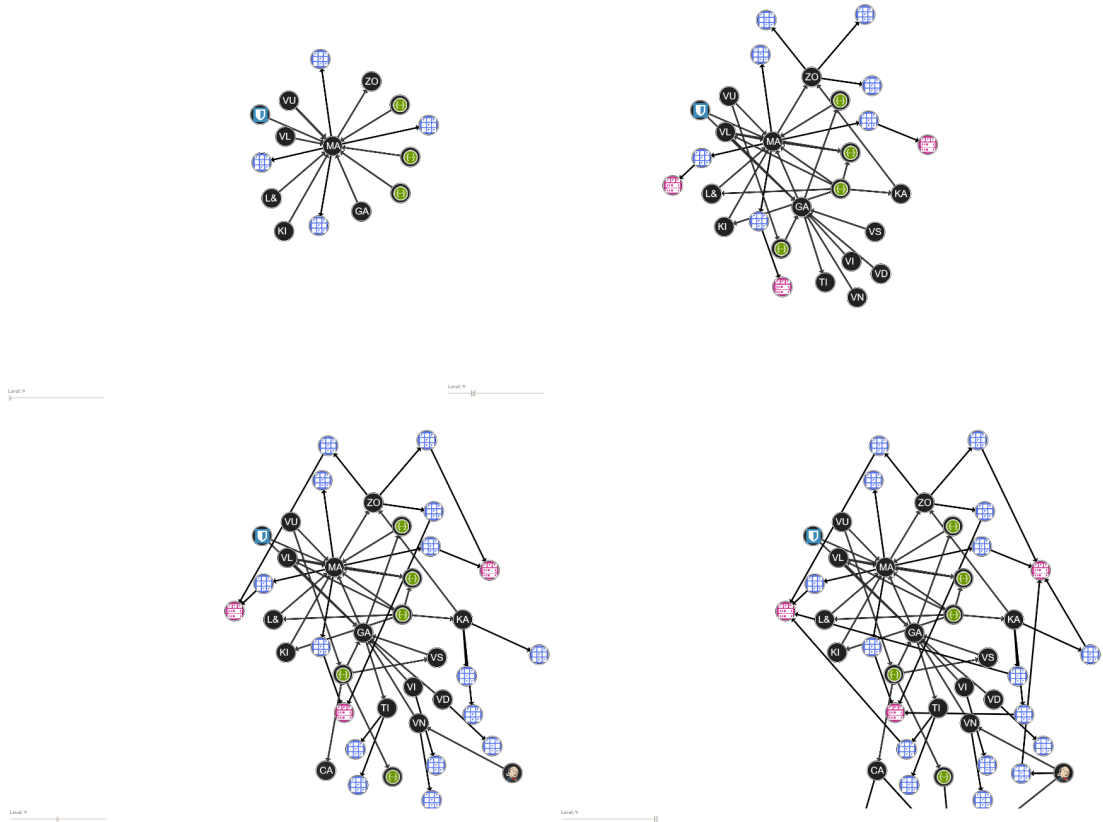


Figure 28: Graph expanding in four levels.

6.2 Adding Features

As there was limited amount of time with the actual user the MVP was used to get the wanted use cases from the users. However to make the most of the meetings with the users some features were developed to help with some of the expected use cases the users might have. So to help the use case of centrality two different methods were developed where the node changes size depending on how many relationships it had connected to itself, see figure 29.

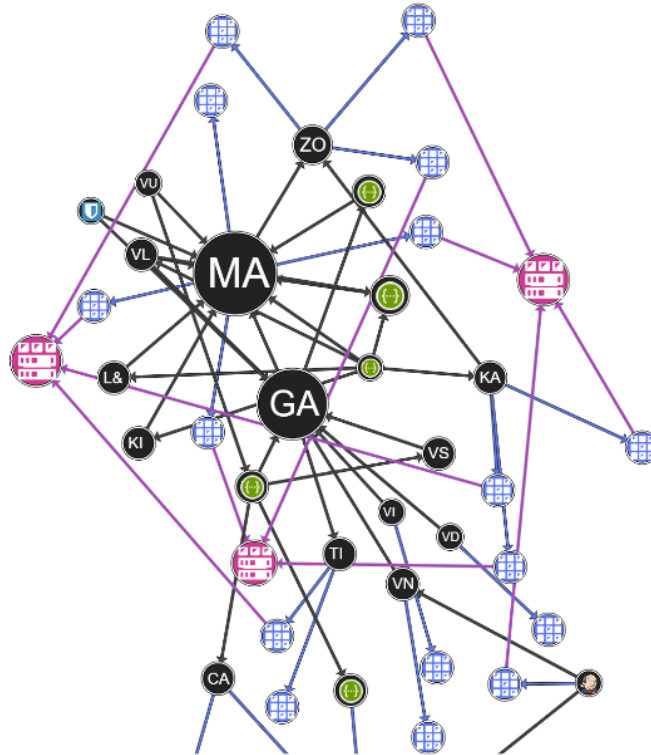


Figure 29: Graph where node size is related to the number of relations to the node.

The first one used the amount of relations that targeted the node and added a fixed size per connection. The other method instead used the PageRank method with every component there was. The PageRank method was added since the first method only showed how many direct relationships it had and does not weigh how 'big' every node is. So that while one node might have more nodes depending on it, it would still appear smaller as it has fewer direct relations targeting it. With PageRank it would look like in figure, 30.

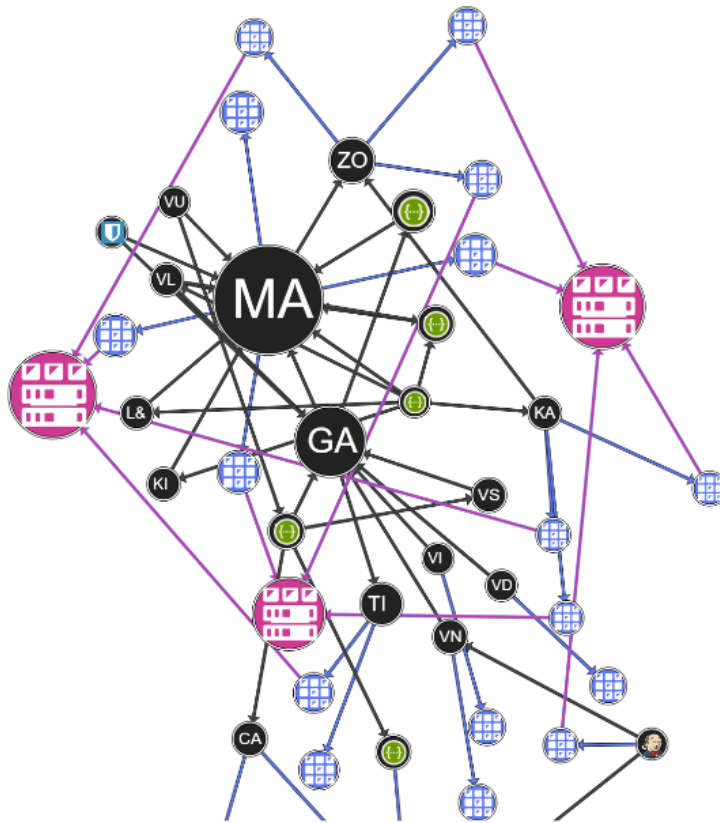


Figure 30: Graph where node size is decided by PageRank.

To make it easier to follow paths, the lines were colored by its source and target node so that crossing lines would be able to be distinguished from each other easier and so that if a line crosses a node it might indicate if it is part of it or just passing through, like in the figures 29, 30. And as a extra feature to fill a specific use case of following relationships that were directed in the same way from the original node, a view like figure 31.

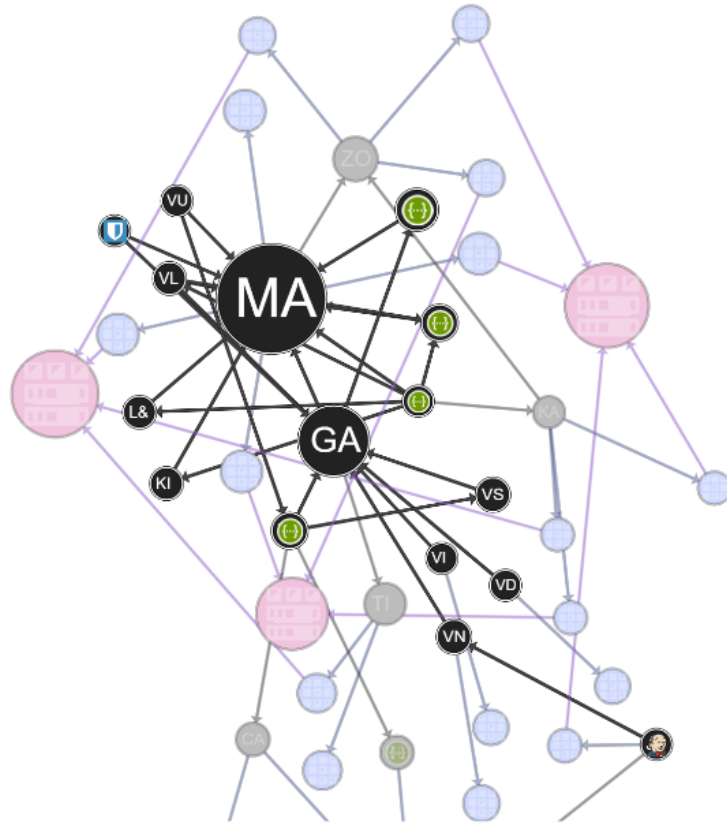


Figure 31: Graph where nodes and links that are dependent of the service Marathon (MA) are shown as usual. Other nodes and links have lowered opacity.

This was meant to also fulfill the path analysis use case. It wont help the user to see lines clearer for everything, but might suggest what can be affected if one node is changed. There was no real feature to help with relationship analysis in this stage however, it was only to be suggested different line patterns or colors to the user.

6.3 Demo Product Notes

During the development it appeared that their own service mapping was wrongly configured. Which became apparent with the amount of relationships shown in the graph. Where services were both dependent on a service as well as that service servers, where in fact they were only dependent on the service which was dependent on its servers. After developing the PageRank algorithm for the node sizes it appeared that it did not show the result that was wanted. Since it divides its PageRank between every component regardless of what kind of relationship

it has. In their own dataset the biggest service divided its PageRank between its servers and a service that it was dependent on. But the servers were just for redundancy, so it does not share the same weight as the service that it needs. This arose the problem of weighting in general which was not implemented in Service Mapping so that relations have different level of relevance. The same goes for the ranking of services, while a bigger service indicates its importance as if it is affected several other things might also get affected. However services that is extra important to the user might be small, so while it looks harmless to change something that it might be a critical part of their operation.

6.4 Setting Up the Test

The purpose of the tests were to find out what use cases are of interest for users and to do somewhat of a usability test of the product. It was an interview divided up into three parts. The first part is asking questions to the user without showing them anything of the visualization, so that we get what the user think they want without any influence from the visualization. The second part is with the MVP where the user gets a bit biased view from the graph and are asked what is missing from this view to figure out what they want from a graph. And lastly they are shown some features that might help some of the use cases. This as a sort of usability test of those features and how useful they are in solving their use cases.

The tests were started by asking for consent and after that they were asked for information about themselves to figure out what kind of role they had, as they can range from upper management with little to no technical experience to the IT technicians that are very technically focused. This was done to categorize user feedback to what the different kinds of users need. Next they were asked whether or not they have used the Service Mapping tool, how much, for what and how it has helped them in their job. If there is any features missing from the tool, if they have felt a need to visualize their Service Mapping and if not, if they think a visualization might have helped them. After that, they were asked what they would use a visualization to do and which types of nodes were of interest for them.

Following that, they were shown a service inside Service Mapping from Infraspight Labs data set. So they would know the data that would be visualized. There after they were shown the MVP displaying what was shown in the service card, which was one degree of separation from the original node. They were then asked what the graph they saw meant, to see if they understood the image. And what they would want to be able to do from here or what they want to know. After that the graph were scaled out to show more and more degrees of separation and ask how the user experiences it and what features they want to have. Next the testee was asked to rank the use cases of path, connection and centrality analysis, and why they chose that rank. And after that they would be shown the extra features and asked how well the features solved the important use cases. And when having seen the complexity of it all, asked whether or not it was important to see the whole network or only the communities like the

one they were shown. Lastly the interviewee was asked if they had any more thoughts or questions.

6.5 Test Results

Five middle aged male test subjects were interviewed, three of them separate and two did the interview together but had their answers noted separately. The interview was conducted using a screen sharing program and the sessions lasted around one hour. The test subjects had different kinds of roles, ranging from more technical to upper management. Some questions were about getting more comparable data and other were more about quality answers. The interviewees answers were noted during the session and later summarized.

6.6 Quantifiable Results

The more quantifiable answers are listed in table 4.

Table 4: The test persons answers.

Test Results							
Test Person	Level of management	Service Mapping Experience	Want a visualization	Why visualization	Ranking (UC)	Features satisfies ranking?	UC4 useful?
A	Mid-lower	High	Yes	UC3, Overview	3>1>2	Yes	No, and not possible
B	Mid-lower	High	Yes	UC3, Overview	3>2>1	No, not UC2	No, and not possible
C	Upper	Medium	Yes	Overview, Tracking	2>3>1	No, not UC2	No, and not possible
D	Lower	Low-none	Yes	UC1, UC3, Overview	1>3>2	Yes	Yes
E	Lower	Low-medium	Yes	UC2, UC3	3>1>2	Yes	does not matter

When asked previously no user seemed to have a good use case for the visualization but everyone wanted one. In this test everyone wanted it and could without showing how anything were setup say in some way what they wanted it for. All but one explicitly described use case 3 (Path analysis) to some extent, where the scenarios were more or less what would happen if something happened to this component. Four of them said more vague things like overview that could mean pretty much anything. Tracking was also suggested by two

of them in different stages throughout the interviews. This is a reference to another tool vScope provides where they keep track of errors and warnings of components, it could also refer to external systems with similar features. One was very interested in relationships, wanting to have easy access to information like how a connection is setup (which is not available in vScope at the moment). And another wanted to use it for use case 1 where you could see what nodes that might have big impact if they are changed in some way. After being shown the MVP they were asked how they would rank the different use cases (1-3).

Use case 1: The second highest in the ranking was UC1 (Centrality analysis) just edging out UC2. For three of them this was important to see what is important to protect, however those that did not think this was important at all either thought that the MVP did a good enough job to show it as is. And one did not think it was that important to just show how many relationships was leading to a certain node. But everyone either wanted the different size feature applied, and/or have it being dependent on a lot of parameters such as importance of the service (no such value in vScope at the moment) or if the aforementioned tracker case shows errors on the node or something.

Use case 2: The lowest rated was UC2 (Connection analysis), where the user that rated it as the most important did not know what kind of information he wanted in relations but he has a lot of experience with different kinds of maps and charts. And he liked arrows are stuff. That user did not use service mapping as much as other users on their company and he guessed that they would be more interested in UC3. The other user that did not put UC2 as lowest rating did not know what it could display at the moment, but would want to be able to toggle on and off information about the relationships (which does not exist in vScope at the moment).

Use case 3: The highest rank by a bit was UC3 (Path analysis). They were all satisfied with how it was handled, using the feature of only showing things that are dependent on a certain node or the other way around (figure 31). As the only use case for path that everyone could think of was when all relationships were headed in the same directions (dependent on something that is dependent on something and so on).

Use case 4: When asked about UC4 (Community analysis) Where you would explore different communities next to each other, then it was requested by one user where they wanted to be able to see the whole network in one picture. Another user was indifferent as they thought that their system was so small that it would be the same thing. The rest of the users did not think it was possible as their network is so vast that it would not say anything at all.

General feedback: As for what components the users wanted to see, everyone wanted to see Services and servers. Other than that it was split. Two had no opinion while others wanted to see an array of things such as licenses, printers, users, owners, ports and suppliers. All of the users expressed a need to themselves chose what kind of nodes would be present.

When the users were shown the graph with only one degree of separation one did not understand what they were looking at, stated lack of descriptive names and labels of the services and relations. There were some confusion from

others as well where they did not know what the arrow meant or where they assumed wrong. But two of the users understood what was going on with the nodes, but had trouble identifying nodes stating that in their vast system a lot of nodes can have the same initials. When expanding it all but one were able to handle two levels of separation, all but one got lost when expanding it further than that, except the user that had a lot of experience with graphs.

6.6.1 Feature Requests

To help with the messiness all users wanted to be able to move the nodes around and place them wherever they want, as well as filter what kind of nodes they want to show. As for helping them understand the context of the situation they want a title in the graph as well and some wanted labels on everything, or that it is at least toggle-able. Everyone wanted to see more of the node name than just the two letters on nodes without an icon. All wanted information given on hover, while they did not know explicitly what information, examples given were name, owner, supplier, tracker case, notes, amount of relations and how critical a component is. Some commonly suggested features are to be able to zoom in and out. Save their views as is. Have a link from the node to their property page in vScope. Integrate the graph with tracker cases. Have the ability to not only start on services but any node in their system to see their relationships. And some features that one or two users requested were dark mode, show things that have something in common, such as the same owner (a bit the same as being able to start on any node). Another request from two users were display everything in a hierarchical manner, so that nodes would be placed in tiers with the same height as the other items in the same tier.

6.7 Test Result Analysis

Some features requested are not possible to complete within the scope of this project, either from not having support for it in vScope, or that it would be too time consuming with the support that currently exists. Every feature that was requested did not affect anything outside the graph, and the only things that would change anything is changing the filters or the placement of the nodes. Otherwise everything is safe to use, and arguably those features would be easy to revert. Some features could be implemented anyway just to show the proof of concept while not having any support from the backend in wait for the actual features. Such as information about the different kinds of relationships, a token text could be added as a feature on hover. While a filter for every kind of component would be very extensive for this project with the current API support it would be possible to implement the feature on the services, servers and datastores that already exists in the MVP. Texts that are labels on the graph like the title and more easily accessible name for every node could also be a possible feature and even toggle-able. The feature to show information on hover could also be added, but since what information it should show will not be investigated further in this project. As the API does not support the saving of

states, this feature will not be implemented. However it will be prepared for the feature regardless. A link to the property page from a select node is possible to create with the resources that exists currently and seems like reasonable feature that will not affect anything else in a negative way. Viewing things that have stuff in common are also descoped, as it is dependent on things other than the aforementioned Services, servers and datastores. The feature to display things in a hierarchical manners will not be done either. The system is very complex and the data is not inherently hierarchical in structure as is, and since the most common relationship by far is between different services it would be extremely bad for some of the quality requirements such as Minimize line crossings and line bending. As users felt that the graph became cluttered and too hard to read, more emphasis could be put on the quality requirements such as symmetry.

6.8 Test Discussion

First of all, if possible it would have been much easier to measure how much the user understood of the test if their own data was used in test, which would have been a logistical problem in this case (as there would need to be a lot of permission settings and such before the tests). These four interview sessions were the only opportunities to test the prototype with the end users, and to get a better understand of how they interpreted the prototype. More test sessions would have been preferable to get more accurate results, however to be given this much time with this many users was very generous. All of the participants were men, which might also affect the result. The product is supposed to be used for both men and women, with different perspectives. Therefore it would have been advantageously to have female participants as well.

Before this project when us sers asked vScopes product owner for the visualization feature, he said that no user had any tangible use cases, just that it would be nice to get an overview. However when asked during the tests everyone but one specifically described use case 3. Use case 1 and 2 were also specifically requested. So as it was assumed that the user would not know what they wanted without showing a MVP seems a bit wrong. It could also just have been enough to give them time to think about what they wanted from a graph and why, as they knew that they would be asked about their opinions in the meeting they had booked. That said, it seemed that the graph helped make the concept easier to grasp what features they wanted and what use cases to focus on. So it would appear that the graph at least helped in some cases here. However for the usability part of the test it would of course not be possible without the prototype. That said, the graph might have been too advanced to be able to see past the design flaws in it or not advanced enough to give the test person a full view of what could be possible in a network graph. But the feedback received seemed to have a good blend of feature request and design critiques, so it would seem to be at an appropriate level. One thing that was noted during the test sessions was that they all requested features and that they wanted to be able to configure what they could see in every feature. The amount of configuration that would be required if every feature they wanted had everything to configure

would be huge. So I think test subjects usually overestimate how much they want to be able to configure. It is a battle between efficiency, learnability, utility and effectiveness.

6.9 Improvements

Several features were made to improve from the feedback of the users. The feathering of the nodes that was used to prevent nodes from ending up on top of each other was removed. Instead a feature was implemented to drag around nodes that placed them where the user wanted them. And to increase symmetry and to avoid the previous problem of figure 26 where there were nodes inside each other, the nodes with the most amount of relations were set in the center instead. This would make it harder to see which node the graph is stemming from. To facilitate context understanding a title was added that said "Relations to and from *Component name*". The name of the nodes were displayed under each node but with the ability to toggle it on and off. The default setting is that the name is displayed for services but not for the servers or datastores. The user would also be able to toggle on and off displaying servers and datastores. This is configured in the improved legend that also shows what the relationships mean. Now to see what kind of relationship two nodes has, the user hovers one of the nodes and all relationships from that node is displayed in text. This text appears in the color of the node that is dependent on the other node. The text is in this version always "Depends on" and angled with the line to give a sense of direction in addition to the arrow. The feature to open a nodes property page inside vScope was also added by holding down the shift key and clicking on the node, which would open the property page in a new tab so that the user does not lose their way in the graph.

Relations to and from Marathon

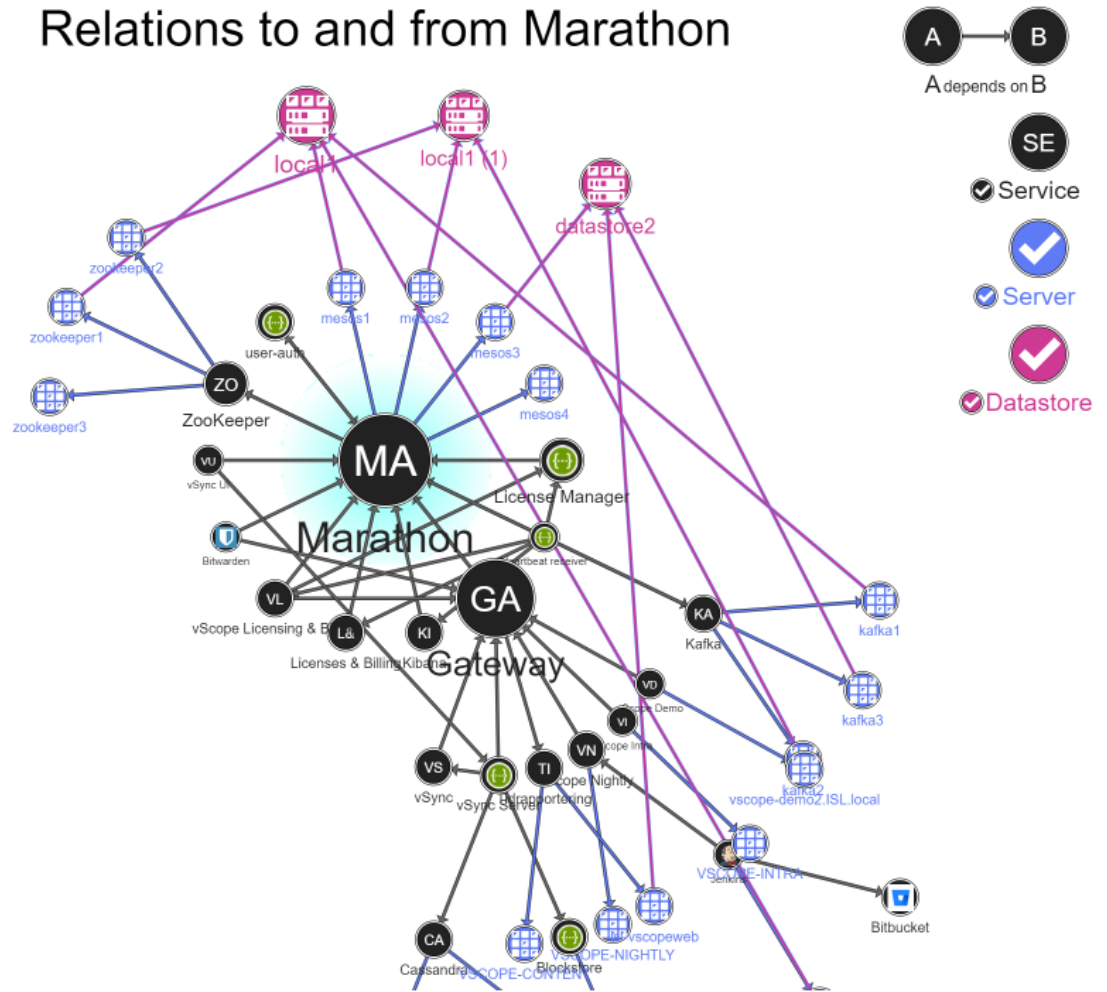


Figure 33: Graph where all nodes are shown as well as their names.

Relations to and from Marathon

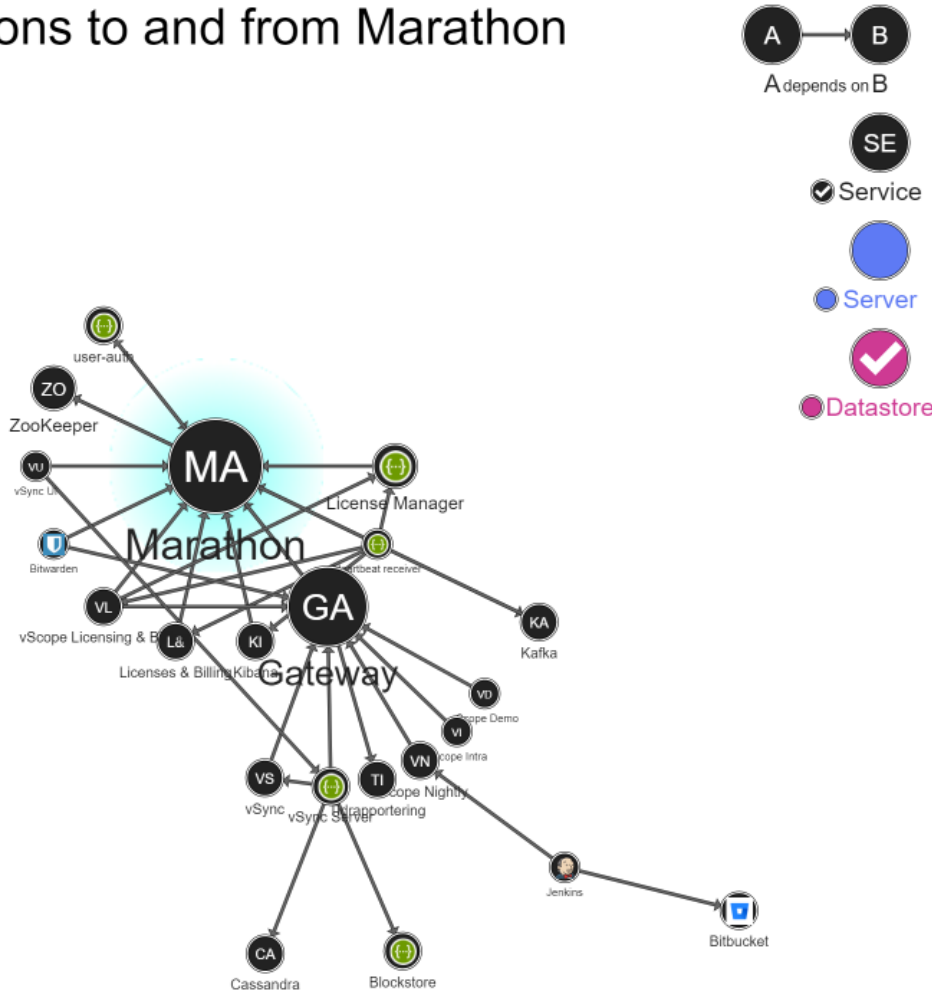


Figure 34: Graph where only services are shown.

7 Discussion

In the visualizations of network context is king. So the product from this project is not applicable for every scenario, but instead it is to guide the process on how to do a network visualization and what to think about when doing so. The product was aimed at the technical networks and the use cases of their users and the nature of their data. With this data it limits the design choices with for example color as the data already has colors and icons assigned to them through Infrasight Labs program. But what the project does is list many tools that the designer has when creating its own visualization and suggestions of what it might be used to do. The project along with that gives an example of an implemented product that has taken these things in mind and that has conducted user tests that show what some users want and what to expect that users might want.

7.1 Use Case Discussion

The use cases discussed in the report seemed to be accurate and what users would want to see would most often fall under one of those. If it was not one of those, it would be overview where it would be that the user did not know what they wanted, but just to see how the system looked. And then the use cases had different levels of importance to the users.

7.1.1 UC1: Centrality Analysis

With UC1 the solution was clear for all the users, but the metrics used might not be the metrics needed. The most important node was displayed as bigger, but the metrics for the most important node was its PageRank, however that does not take into account how important relationships or the nodes were so it might be misleading. In a case from Infrasight Labs, one service runs on four servers, but the four servers works as a redundancy so that if one or two servers breaks down it wont affect the service as a whole. But that relationship is weighed the same as any other relationship, so the PageRank might be misleading. However there is not any information about the importance of the relationship at the moment. The same is for the components themselves, different services are of different importance to the company but no such information is to be displayed. So while the users were satisfied with the solution and understood it, it does not necessarily show the 'complete truth'.

7.1.2 UC2: Connection Analysis

UC2 was not heavily focused as there was no real good information to display. Instead only a proof of concept was implemented but not tested, where on hover a token text was formed. This as it was not prioritized by users and by the ones who wanted it they did not know what to display. But with further investigation and more data available this could be explored further.

7.1.3 UC3: Path Analysis

With the test results it would seem that while the graph itself was not good enough to solve the most important use case, UC3, easily without interaction, the additional features were. As when the data sets become big enough it starts getting cluttered, so while it would have been more efficient from a usability perspective to design it in such a way that the user would not have to interact with the graph to get the information they want. So it could be improved with user customizable filtering, some of which were implemented, but a better placement algorithm for the nodes might have solved the problem even better. The solution is fine within the context of the program as it can tailor a bit to the user, but might be considered worse if the user wants to share their view with others as it is dependent on interaction.

7.1.4 UC4: Community Analysis

Community analysis is sort of what is happening, as the graph shows one community. However to show the community in relation to other communities were not requested by most of the users just because of how vast their data sets were. The program however has only been tested on Infrsight Labs own data set, and to test it for other sets would be very useful for quality reassurance. Also more thought in how the user should operate the graph would benefit it, as of now features are there more as a proof of concept. And to implement the descoped features would probably improve the product further.

7.2 Work Process Discussion

The work process was not always optimal as there was limited amount of time to interact with the users, and to court the users it was elected to not test low-fi prototypes as one usually does in a design process, and instead show the MVP. However with the way it transpired I do not think that it would improve the design process significantly. However that might have been due to a lucky guess with what the users actually wanted, which was prepared for in the MVP. However this 'lucky guess' could very well be from the thorough investigation process with the existing products and the theory. Of course with more time the improvements made would preferably be tested on the users. And the descoped features would be investigated more thoroughly. Preferably a more diverse range of test subjects that still are users would be tested, but to even been given so much time with relevant users was extremely helpful and contributed a lot with the process.

8 Conclusion

Visualization of data is a great way to convey information to an audience. When visualizing data it is important to pick the right medium for the data set, and what the audience needs to know from the information. For small data sets there are several types of graphs that are viable, and one might pick whichever one likes best, and focus on aesthetics to make the graph more enjoyable. But with sizeable data sets it would be of greater importance to focus more on the quality design features of the graph. In bigger data sets, network graphs becomes the most viable option due to its ability to scale and its abilities to fulfill qualitative and functional requirements.

When visualizing, there are several tools to use (such as use of color, size, placement and interactivity) that will help users with their analysis of the data. It is important that the creator of the graph understands their user's needs, the data and their tools. The nature of the data might change how you can use different tools, so while you might want to use colors to signify for example communities or similar, colors might already be a part of what makes a component recognizable to a user. While one would prefer that a user would understand the graph and fulfill their use cases by just looking at the graph it is also possible to implement interactive features that help manage the information or that does analyses for the users.

While the users were interested in customizability of everything in the graph, it might make the learnability worse but increase the utility and efficiency of the system. So a balance of how much to customize needs to be reached. Customizability is important since while the users might be from similar backgrounds and have similar use cases, they are most often not a monolith. So they might have different points of interest in the data, and different levels of understanding. It is also important to see what the users understand. It can not be assumed that the user understand icons and colors even though they are frequently used in the system. Therefore, names and legend might have to be used to clarify different aspects.

In summary, to efficiently visualize a network it is important to consider and understand the nature of the data, the visualization tools and the users.

9 References

- [1] "Six Degrees of Separation". Wikipedia, *Visited February 2020*. https://en.wikipedia.org/wiki/Six_degrees_of_separation
- [2] "Wikipedia:Wiki Game". Wikipedia, *Visited February 2020*. https://en.wikipedia.org/wiki/Wikipedia:Wiki_Game
- [3] "Your IT environment – All in one place". Infrasight Labs, *Visited February 2020*. <https://www.vscope.net/>
- [4] Preece, J, Rogers, Y, Sharp, H (2002). "Interaction Design: Beyond Human Computer Interaction" John Wiley & Sons, Inc.
- [5] "Aesthetics". Interaction Design Foundation, *Visited November 2019*. <https://www.interaction-design.org/literature/topics/aesthetics>
- [6] "Data Visualization for Human Perception". The Interaction Design Foundation. *Visited February 2020*.
- [7] "Gestalt principles". Interaction Design Foundation, *Visited October 2019*. <https://www.interaction-design.org/literature/topics/gestalt-principles>
- [8] Vale, F. Wayson, T. (2018). "Charts and custom visualizations - beyond the map". esri, *Visited November 2019*. <http://proceedings.esri.com/library/userconf/devsummit18/papers/dev-int-101.pdf>
- [9] Wilke, C. (2019) "Fundamentals of Data Visualization". O'Reilly.
- [10] Goksör, N., Kemvik, A (2019). "Data Visualization of Product Relations - An Interactive Virtual Reality Solution" Lunds Tekniska Högskola.
- [11] "Graph database". Wikipedia, *Visited October 2019*. https://en.wikipedia.org/wiki/Graph_database
- [12] Shetty, S. (2017). "When, why and how to use Graph analytics for your big data" Packt Hub, *Visited November 2019*. <https://hub.packtpub.com/when-why-and-how-to-use-graph-analytics-for-your-big-data/>
- [13] M. A. Porter; J.-P. Onnela; P. J. Mucha (2009). "Communities in Networks". *Notices of the American Mathematical Society*. 56: 1082–1097, 1164–1166.
- [14] Sugiyama, K., S. Tagawa, and M. Toda (1981). "Methods for visual understanding of hierarchical system structures." *IEEE Transactions on Systems, Man and Cybernetics*.
- [15] Sindre, G., B. Gulla, and H. Jokstad (1993). "Onion Graphs: Aesthetic and Layout." *Proc. 1993 IEEE Symposium on Visual Languages*.

- [16] Purchase, H. C., R.F. Cohen, and M. James (1996). "Validating Graph Drawing Aesthetics." Proc. Symp. Graph Drawing
- [17] Aris, A. Shneiderman, B (2006). "Network Visualization by Semantic Substrates". IEEE Transactions on Visualization and Computer Graphics.
- [18] Ware, C., Helen Purchase, Linda Colpoys, and Matthew McGill (2002). "Cognitive measurements of graph aesthetics." Information Visualization.
- [19] Grant, R (2018). "Data Visualization Charts, Maps, and Interactive Graphics". Chapman and Hall/CRC New York.
- [20] Gleich, D (2015). "Pagerank beyond the Web". SIAM.
- [21] "PageRank". Wikipedia, *Visited January 2020*.
<https://en.wikipedia.org/wiki/PageRank>
- [22] Ribecca, S. "The Data Visualisation Catalogue" The Data Visualisation Catalogue, *Visited October 2019*. <https://datavizcatalogue.com/>
- [23] "Visualizing Graph Databases with KeyLines". Cambridge Intelligence, *Visited November 2019*.
<https://www.youtube.com/watch?v=9EB6cZjT4nQ>
- [24] "ConnectTheDots: Les Misérables Character Co-occurrence". Databasics, *Visited November 2019*.
<https://www.databasic.io/en/connectthedots/results/5b736b0c9dfb0a00e15f0816>
- [25] "Alchemy.js". GraphAlchemist, *Visited November 2019*.
<http://graphalchemist.github.io/Alchemy/#/examples>
- [26] "Javascript Fatigue Kumu". Kumu Inc, *Visited November 2019*.
<https://Kumu.io/dan/javascript-fatigue#javascript-fatigue/nodejs>

A Appendix

Here are test summaries from the different test persons in Swedish.

A.1 Test Person A & B

De hanterar och dokumenterar systemen. Båda är lite blandad nivå i tekniken. De sitter i en driftgrupp senast. Har hand om system, nät och allt möjligt.

De har haft det länge, men började dokumentera allt för ca: 2 månader sen. Men allt är inte ifyllt än. De har påbörjat relationskopplingar osv.

De använder det för att byta ut inventeringslistan och koppla hur saker funkar. Inventering, dokumentation. T.ex AD, licenser, klienter, servrar och skrivare. De inventerar inte nätverket så de försöker hålla koll på relationerna. Samlad bild på användare och servrar och se hur de är beroende av varandra.

De vill ha en övergripande skiss så att de kan dela med det med andra. Andra kan se servicekort men inte förstå meningen i det. Både på avdelningar som inte har något koll på it, men även för nya folk i It-avdelningen. De vill kunna ha en uppdaterad dokumentation som de kan dela med sig av. Så de vill få service mapping att gå mer från it till andra avdelningar.

De vill kunna sortera på flera saker och lättare behörighetsdelning (inte relaterat till exjobb).

De tror att det hade kunnat ge en snabb överblick till helpdesk till exempel. De har fått information om en incident och se vad det kan ha påverkat eller vad som påverkat det. (Use case 3)

De hade velat använda det för att se kopplingen mellan olika system och funktioner. Flöde, felsökningsperspektiv. Planeringsarbete. Och de vill starta om detta, vad kommer det drabba. (Use case 3)

De vill se komponenterna webserver, application pools, sites, services, servrar/maskiner, databaser. I kartan. Och på något sätt se grupper, ägare, leverantör.

När de blev visade prototypen så trodde (och hade rätt) person A att de förstod vad pilarna betydde men person B tyckte det var otydligt.

Person A ville att man skulle kunna visualisera saker mer hierarkisk. Båda ville kunna hoovra för information och se fullt namn på allting man kan få plats med. Det funkar inte att bara ha de två bokstäverna som finns annars. För att det ger bättre överblick.

De var intresserade att se längre än första steget. På andra steget tyckte de att det blev mer plottrigt (det blev lite fel i testet här med en del buggar). Person B trodde det kunde vara ett bra verktyg för att se ifall man dokumenterat rätt eller inte.

De hade velat kunna filtrera bort saker. De tycker att saker borde försvinna då som är endast kopplade till den. Vilja se nära relaterade åt ett håll (use case 3). Svårt att följa vägarna som de är nu.

Person A rankar 3;1;2. 3: Samma som innan, tycker det är svårt i detta laget, men funktionen uppfyller A's krav. 1: Har lite svårt att säga varför, men vill kunna se viktighet i andra saker som t.ex tracker case eller liknande. 2

Person B rankar 3;2;1. 3: samma som A. 2: Vet inte vilken kopplingsinformation som finns i vScope atm, men tänker om man på något sätt kan sätta på/av kopplingsinfo i relationer. Vill ha information som troligen inte finns i vScope just nu. 1: Känner att det kan hjälpa men vet inte när.

Båda featuresen hjälpte med use case 3 och 1 och störde inte. Men ser inga fall man tjänar på olika storlek på noder.

De vill se andra features som att man kan se vyer med gemensamma förvaltare.

De tycker det är viktigt att den endast visualiserar ner i hierarkin och inte upp igen.

De tror att deras dataset är för stort för att ha en bild av allting.

De tror att en visualisering hade hjälpt dem absolut, de har brist av systemskissar.

Övrigt: De hade velat ha rubrik så man vet var man är. Sätt att exportera det (inte relevant för exjobb). Sätta på och av namn för allt. Ha en länk från grafen till egenskapssidan för komponenten. Person A vill ha Darkmode. Integrera med tracker case (funktion som håller koll på varningar och errors till komponenter). T.ex visa det i komponenterna. Visa saker som har något gemensamt, t.ex samma ägare eller user. Kunna gå in på specifik dator och se vem som äger den.

A.2 Test Person C

Hen är en enterprise-arkitekt. Håller ihop helheten på företaget. Ser relationer mellan teknologier. Lite mer upper management.

Hen har använt service mapping lite tidigare, men det är mer andra som använder det oftare på företaget. Hen har använt det för att mappa tjänster, men använder oftast ett annat system för att visa det. Hen kvalitetsgranskar information som är kopierad från deras masterdata som finns på ett annat ställe. Som inte har inventerad data.

Hen saknar logisk gruppering av tjänster i service mapping.

Hen kände att en visualisering var inte nödvändigt för Hen utan för andra i företaget, de som använder det oftare. Vill se antal servrar, varningar, problem med servrar och integration med andra tjänster de har som håller koll på ändringar osv i visualiseringen.

Hen hade velat ha en visualisering för att skapa förståelse så att folk slipper ställa frågor om allt om t.ex vilka servrar/OS som behöver uppdateras. Samt velat ha en geografisk karta för att se var infrastrukturen är fördelad.

Har ingen egen åsikt om vilka komponenter som ska visas, tror att olika delar av organisationen vill ha sina egna saker.

Tycker att det är otydligt vad pilen representerar, om det är dataflöde eller dependencies för att det är olika beroende på system annars. Svårt att se utan kontext. Hen vill att jag ska namnge Vyn. Ha legend som visar vad pilar och symboler betyder. Samt att man inte ska blanda beroende och flöde, vilket inte görs i bilden.

Samma typ av vy i större kontext med fler komponenter, vill kunna sortera noder genom att dra i dem, samt att filtrera och zooma in och ut.

Hen vill ha information vid hover, viktiga bitar som namn och notes, vad folk än känner är relevant.

Hen vet vad allt är så hen tyckte det var begripligt. Vill fortfarande kunna zooma och filtrera, vill kunna byta vy så att man bytar vad som sätts i centrum. Så byter man nod så byter man vy.

Hen säger att hen tror folk tycker väg är viktigast och sen vad som är viktigast är väg- ζ relation - ζ viktighet. Hen tycker att vilken sorts relation är viktigast för Hen. Att man isåfall har en legend med olika sorters pilar som betyder olika saker. Men kan inte säga vad det ska vara för sorters relationer. Hen missförstod min förklaring av väg, men tyckte att vad som skulle vara viktigt är att se är det som är gjort i funktionen där man endast ser ett beroendehåll från en nod. Kanske se alla kopplingar mellan två noder, men tror inte att det är relevant så länge de inte har samma sorts kedja (Alla relationer går åt samma håll). Hen tycker att man kan se vägen, men man får inte informationen så lätt. Måste anstränga sig. Hen tycker att det är redan självklart vilka som har flest kopplingar till sig, men gillar koncept som PageRank, men att man har fler faktorer i det, som hur viktigt det är eller tracker cases (Verktyg i vScope som håller koll på status). Tyckte dependencies featuren var bra och att det blev lättare och trevligare att titta på den grafen och att det hjälpte med storleken.

För att visa relationer vill man inte skriva information bredvid för att det blir för plottrigt, men kanske vid hover.

Hen tror att det kunnat hjälpa hen i arbetet, men främst förståelse för tjänstmanagers så att det blir lättare att förvalta tjänster, så de vet var de ska gå med saker och hur de ska förvalta tjänster. Att visa hela nätverket utzoomat hade inte funkant alls på ett företag i deras storlek. Det hade varit aldeles för stort.

Hen vill att man ska kunna gruppera om så att det fastnar, så man kan öka tydligheten.

A.3 Test Person D

Hen är en it operations manager som ansvarar för allt som har med teknik att göra och drift. Avtal med mera. Hen hade inte använt service mapping men fått det demat för sig, Hen tänkte börja använda det för att ersätta befintliga excellark så att de kan öka transparensen mellan avdelningar. Vilka applikationer de har så att t.ex folk inte kommer och köper in system som redan finns osv. Hen har känt ett behov av att visualisera det innan. Hen hade velat ha en systemkarta och se relationerna. Hen hade velat se servrarna och dess relationer, information om dem och uttryckte egentligen både use case om väg samt om storlek. Ett use case var att Hen ville se hur en förändring skulle kunna påverka andra komponenter. Samt se vilka saker som man inte borde pilla på om man vill undvika förändringar i många saker. Hen vill se servrar, mer kan Hen inte säga. Hen har inte satt in sig i service mapping än så. Vad händer vid en förändring. Och en korrekt dokumentation över systemet. Hen hade velat kunna se bättre bilder, tjänsterna säger Hen inget, Hen vill ha en rubrik, Hen vill se vilken tjänst Hen utgår ifrån och vad det är för karta Hen

vill titta på. Hen vill kunna dra och släppa nodes och spara dessa vyerna till senare. Hen tycker andra iterationen är rörig, massa pilar och Hen vill dra i lite saker. I detta skedet så saknar Hen fortfarande namn, Hen vill ha namn på alla komponenter alltid eller val att filtrera bort det. Samt check boxes för att dölja saker Hen ej vill se. När det sen var helt utfällt så tyckte Hen att det var ännu stökigare, annars samma feedback. Hen vill att ifall en relation är åt båda hållen så vill Hen se det. Hen vill kunna dölja allt som inte är i direkt relation med komponenten. Hen vill även kunna se hela kartan utzoomad så även andra communities är med för att lättare kunna se var man kan pilla om man vill. Rangordning. 1: Sa det innan att Hen ville ha det, men viktigaste komponenter vill Hen se lätt. 3: Ville kunna se en saks närmre relationer (antar att det betyder att Hen ville se ett relationsflöde som dep/isdep). 2: Vilken sorts relation verkade inte vara så viktig änsålänge. Hen kunde gissa vilka relationer som var viktigast på ordinariebilderna men tyckte det blev tydligare i unstupify. Hen trodde att pilarna betydde dataflöde och inte beroende till varandra. (Så de är på fel håll). De nya funktionerna hjälpte till "enormt". Hen ville kunna släppa in och uppdatera alla sina systemkartor, bra att Hen kunde slippa in på shellpoint osv ser fram emot att använda visualiseringen sen. Hen bad om att kunna skapa egna vyer eller kartor och även kunna lista saker om den markerade komponenten.

A.4 Test Person E

Hen är samordnare för IT-infrastruktur. Samordnar server, lagring och datakommunikation. VMware, nätverkskommunikation och AD. Hyffsat lågnivå, alltså väldigt teknisk. Har kollat på service mapping innan, testat det när det utvecklades. Planer att snart fylla i det och byta ut excell därifrån. De använder det för att se systemägare och förvaltare och deras kopplingar. Knyta servrar osv med certifikat. Så knappt börjat med det.

Hen saknar visuella relationer så att Hen kan se beroende enkelt. Hen vill se vad som händer ifall man gör något på en server. Hen vill kunna lägga till och ta bort information i table explorer (inte relevant för mig). Vill ha vy som de har i excel och kunna filtrera efter ägare t.ex. Vilka system som ägs av vissa förvaltningar och vilka kolumner som ska visas.

Hen vill ha mer information om hur de ansluts, t.ex de kopplas genom denna porten. Vill kunna se tcp-portar osv. (Funktionen saknas i vScope) Hen visste inte om featuren att deklarerar olika sorters beroende.

Vill se det visuellt men vet inte helt varför mer än att se relationer mellan objekt och en visuell bild av systemet i sig. Men kallar det en nice to have.

Hen hade använt en graf för att kvalitetssäkra förändringar så att de vet vilka saker som kan påverkas vid en ändring av en viss komponent. Så se vägar, aka use case 3.

Hen vill se tjänster, servrar, konton, portar, ägare, förvaltare, leverantör, support. Kanske att man kan expandera. (Vill ha det lättare att inventera från AD till grupper och users i service-kortet. Utanför exjobbet)

Hen förstod de olika ikonerna och fattade hur beroenden var riktade. Vill Hoover för snabbkort med vissa detaljer, namn, description, samt kunna själv välja vad Hen vill visa där. Hen vill se en snabb överblick som trackercase vid Hoover. Vill se skillnader på olika sorters connections, indikation på relation till certifikat och kunna hämta certifikat för att visa dess relationer istället för att utgå ifrån en tjänst. Så att man gå från vilken nod som helst för att visa vad det är relaterat till.

Hen tycker det är svårt att se vad som är tjänster så att kanske byta det till en annan form för att urskilja det. Lite svårt att se relationer. Hen vill kunna ha ett filter så Hen kan sätta på och av olika komponenter samt dess relationer. Ha det i kanske en lista. Hen vill se user-groups och kanske antalet users. Hen vill kunna hålla på en pil för att se connectionen. T.ex vilken port de pratar över (funktion saknas i vScope, men kanske skriva description eller typ av kommunikation). Hen vill veta dataflöde och vad den gör i den här kopplingen (Finns inte i vScope). Hen är intresserad av en hierarkisk struktur för att lättare hålla koll på servrar osv. (Kanske påverkat av ett tidigare existerande verktyg).

Rangordnar det som 3,1,2. Hen vill se vad som påverkas i en ändring, kanske i textform på tjänstekortet (men var lite osäker ifall Hen bara ville ha en nivå där eller flera).

Vill se hur viktade beroende är (Funktion saknas i vScope), samma sak med hur viktiga olika tjänster är. (Funktion saknas i vScope).

Hen vill kunna välja olika sätt att visa saker, som t.ex markera hur långt ifrån originalnoden de är genom storlek eller något. Vill se antalet relationer vid Hoover.

Det var svårt att se relationsvägen från början, men blev lättare vid featuren att se beroende.

När jag ändrade storleken så vill Hen själv välja vad som ska indikera en storleksökning, t.ex antal relationer in, eller deras vikt, eller vikt av noden osv.

Hen kunnat hjälpa men Hen tyckte att det var stökigt i slutet oavsett.

Hen hade kanske velat ha systemet i en dashboard men att det är mer nice to have. Eller att visa det med tracker case så att i content ha versioner där man kan visa vad som påverkas när något slår larm (Inte relevant för exjobbet).

Så att community inte hade varit relevant för hen eftersom det hade allt hängt ihopa.